Éléments d'informatique – Cours 9. Fonctions (2)

Pierre Boudes

22 novembre 2011

Types

- Types en C et entrées/sorties associées
- Conversions automatiques entre types

Pile d'appel

- Rappel sur les fonctions en C
- Traces et mémoire
- Pile d'appel

Utiliser les fonctions d'une bibliothèque

Longue démo (menu)

Types en C et entrées/sorties associées

Type des caractères `char` :

- Déclaration et initialisation : `char c = 'A';`.
- Représentation sur 8 bits, ASCII, ISO-8859-x, UTF-8.
- E/S : `%c`.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Source : Wikimedia Commons, public domain.

Conversions automatiques entre types

Conversions automatiques entre types

- Sans changement de représentation :
  - `char` vers `int`
  - `int` vers `char` (troncature)

Conversions automatiques entre types

- Sans changement de représentation :
  - `char` vers `int`
  - `int` vers `char` (troncature)

```
char c;
int n;

n = 'A' + 1;
c = n + 24;
```

Conversions automatiques entre types

- Sans changement de représentation :
  - `char` vers `int`
  - `int` vers `char` (troncature)

```
char c;
int n;

n = 'A' + 1;
c = n + 24;
```

- Avec changement de représentation :
  - char ou entiers vers réels
  - réels vers entiers ou char

```
double x;
int n;

n = 3.1;
x = n;
```

Conversions automatiques entre types

- Sans changement de représentation :
  - `char` vers `int`
  - `int` vers `char` (troncature)

```
char c;
int n;

n = 'A' + 1;
c = n + 24;
```

- Avec changement de représentation :
  - char ou entiers vers réels
  - réels vers entiers ou char

```
double x;
int n;

n = 3.1;
x = n;
```

Rappel sur les fonctions en C 

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, **espace mémoire**)

Rappel sur les fonctions en C 

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, **espace mémoire**)

Voyons de manière plus précise cette question d'espace mémoire.

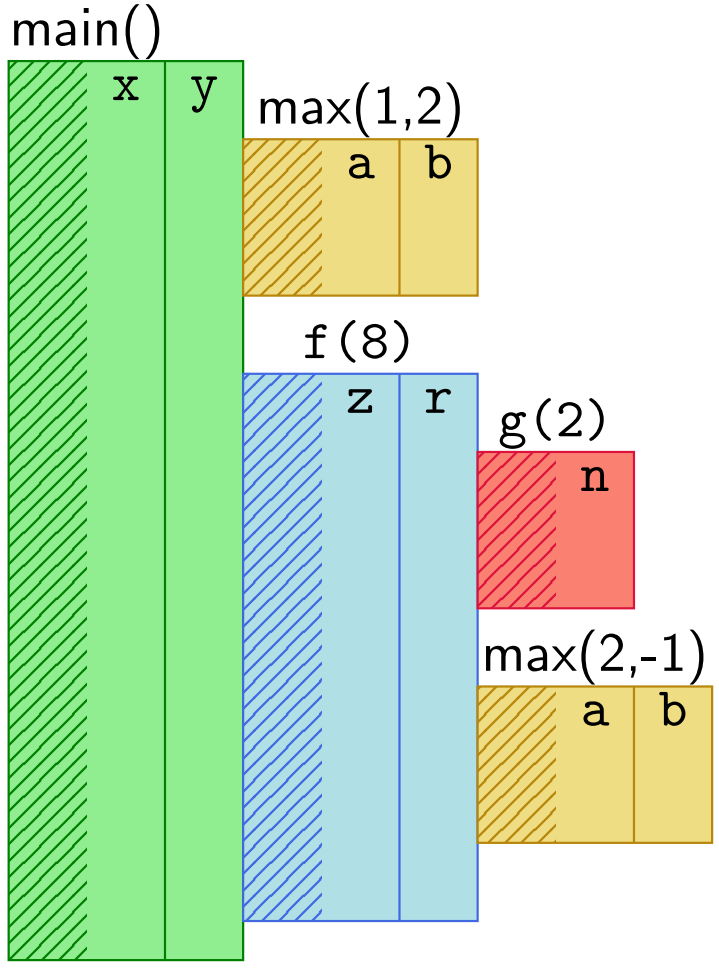
Traces et mémoire 

Rappel : nous faisons la trace de chaque appel de chaque fonction que l'on a défini (pas les fonctions externes, comme printf).

Traces et mémoire

Rappel : nous faisons la trace de chaque appel de chaque fonction que l'on a défini (pas les fonctions externes, comme printf).

La trace d'un programme donne schématiquement ce type de dessin

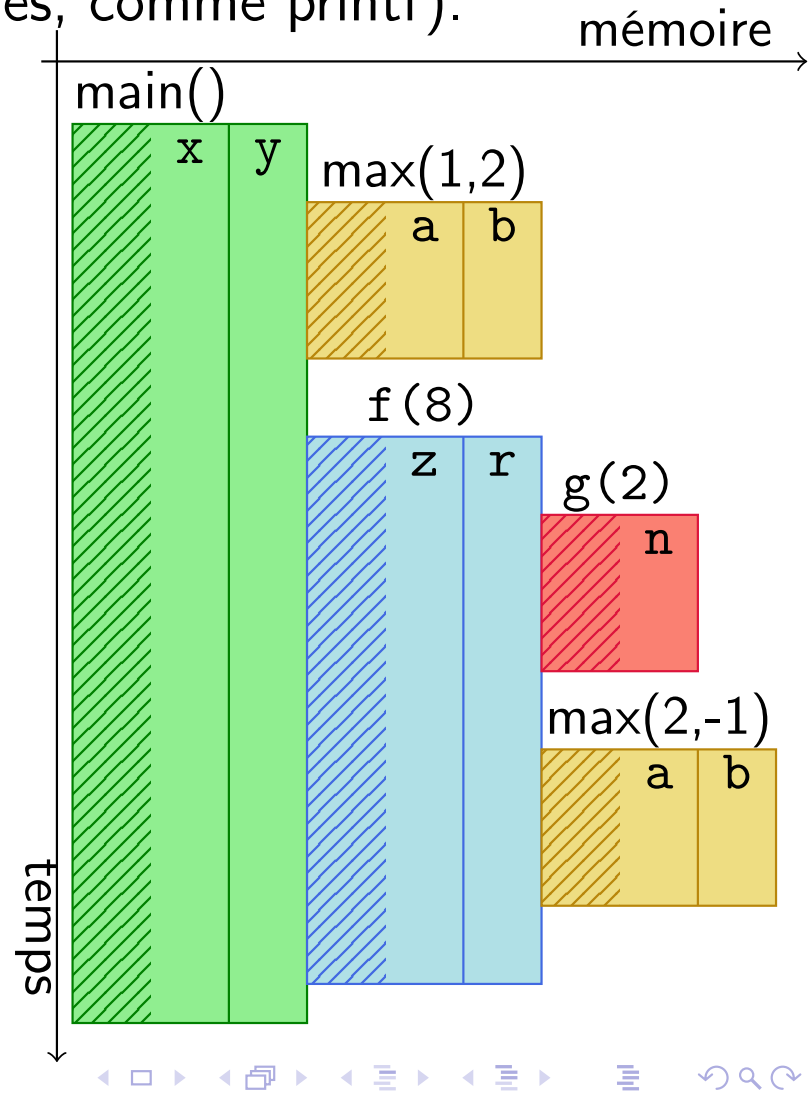


Traces et mémoire

Rappel : nous faisons la trace de chaque appel de chaque fonction que l'on a défini (pas les fonctions externes, comme printf).

La trace d'un programme donne schématiquement ce type de dessin

- verticalement, c'est le temps
- et horizontalement, l'occupation mémoire

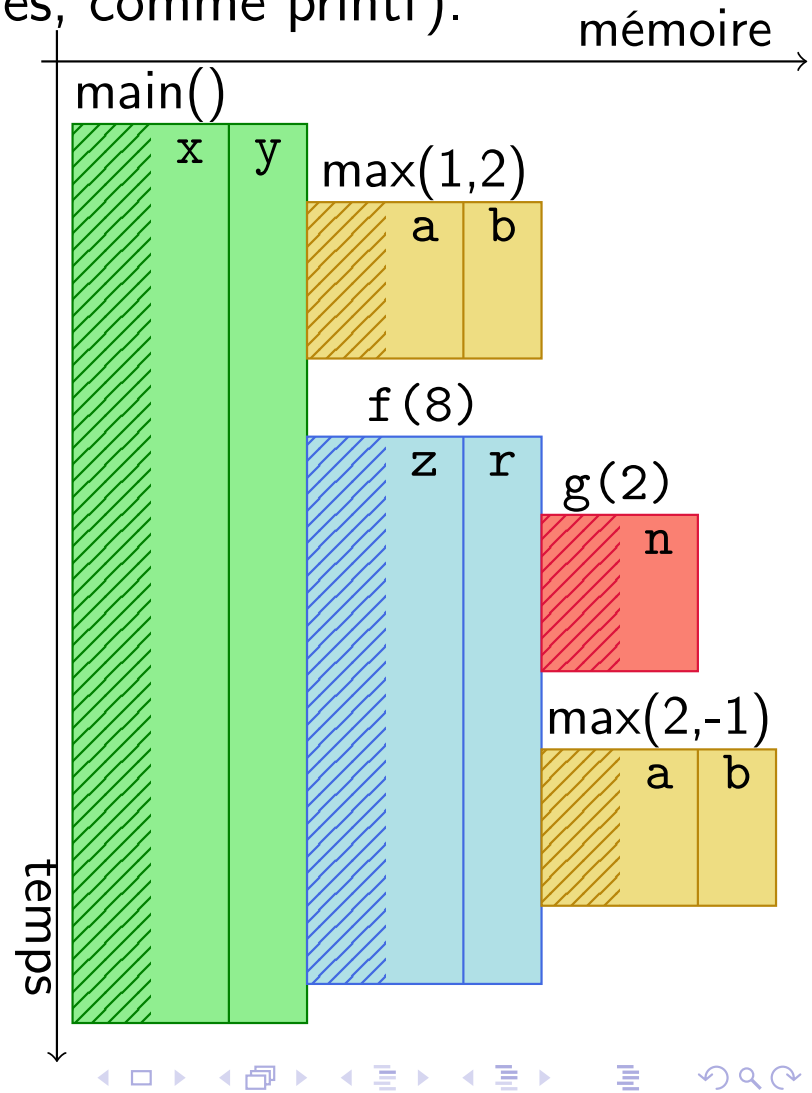


Traces et mémoire

Rappel : nous faisons la trace de chaque appel de chaque fonction que l'on a défini (pas les fonctions externes, comme printf).

La trace d'un programme donne schématiquement ce type de dessin

- verticalement, c'est le temps
- et horizontalement, l'occupation mémoire
- un appel de fonction occupe une portion de mémoire, puis la libère.

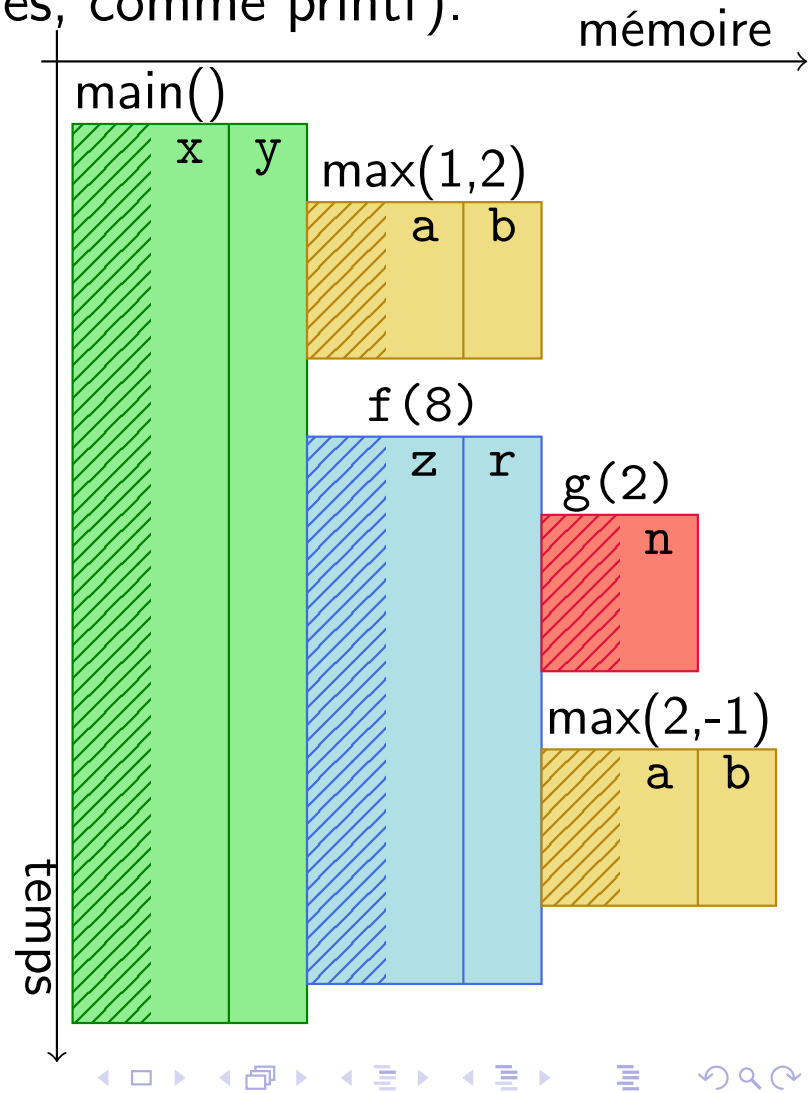


Traces et mémoire

Rappel : nous faisons la trace de chaque appel de chaque fonction que l'on a défini (pas les fonctions externes, comme printf).

La trace d'un programme donne schématiquement ce type de dessin

- verticalement, c'est le temps
- et horizontalement, l'occupation mémoire
- un appel de fonction occupe une portion de mémoire, puis la libère.
- la trace représente réellement ce qui arrive dans vos programmes.



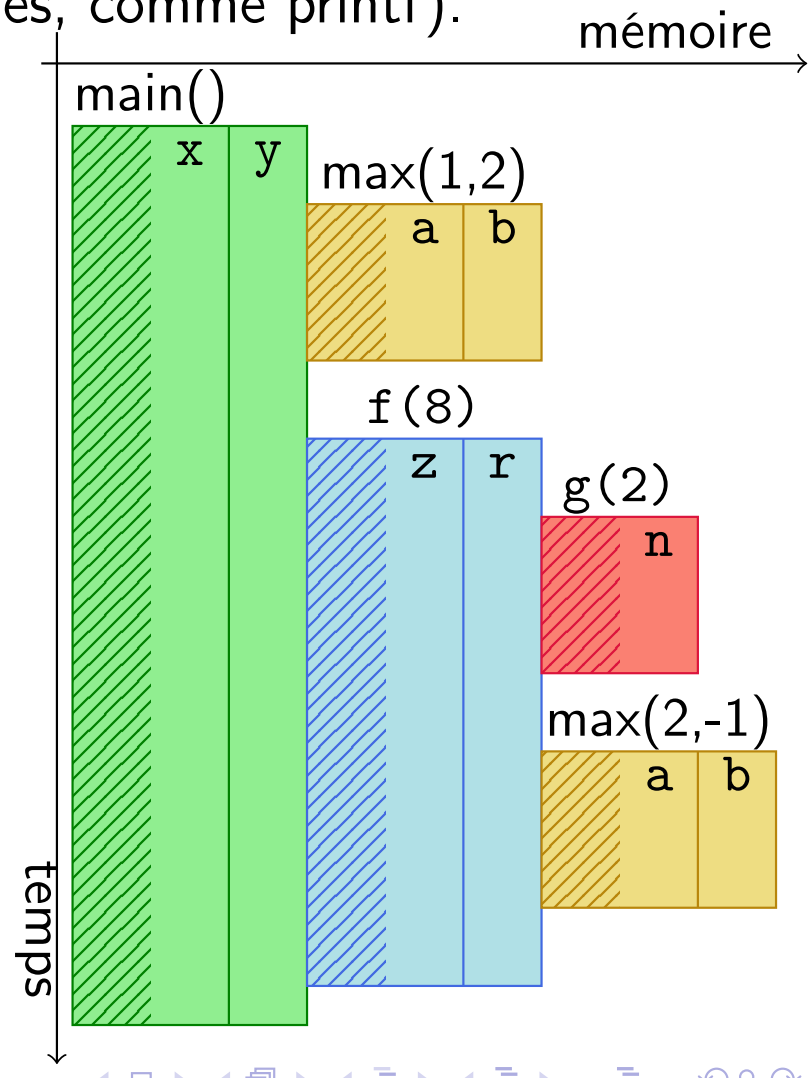
Traces et mémoire

Rappel : nous faisons la trace de chaque appel de chaque fonction que l'on a défini (pas les fonctions externes, comme printf).

La trace d'un programme donne schématiquement ce type de dessin

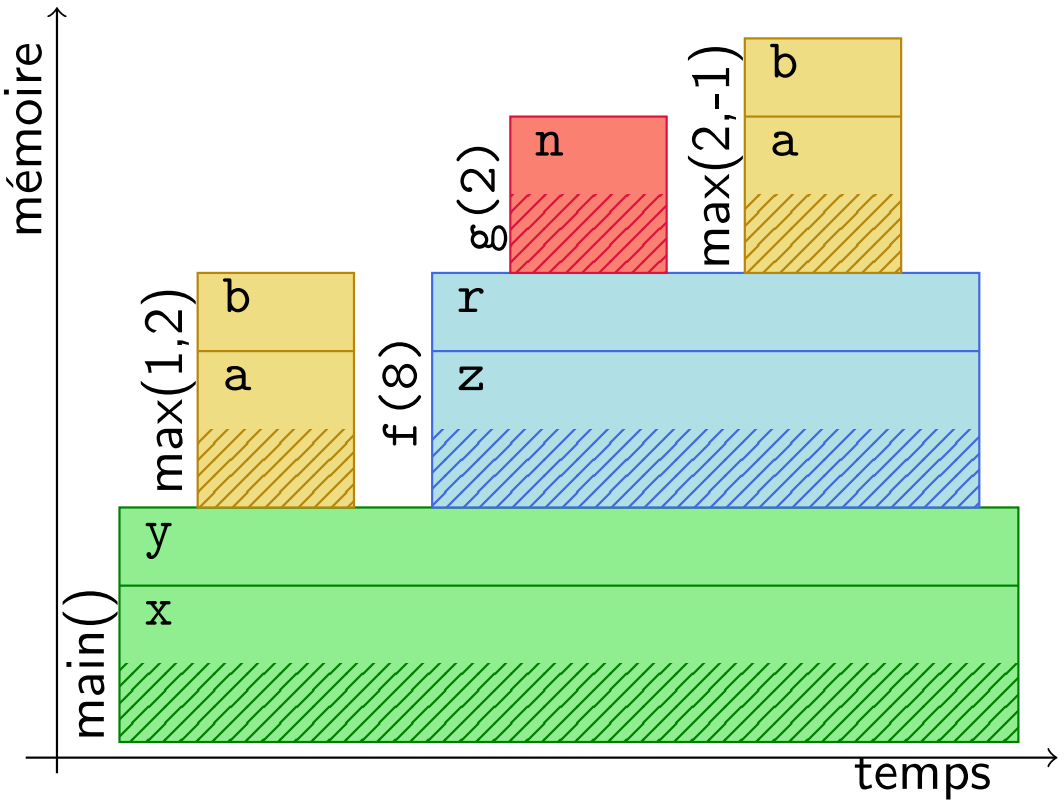
- verticalement, c'est le temps
- et horizontalement, l'occupation mémoire
- un appel de fonction occupe une portion de mémoire, puis la libère.
- la trace représente réellement ce qui arrive dans vos programmes.

Un appel de fonction peut-il modifier la mémoire d'une fonction appelante ?



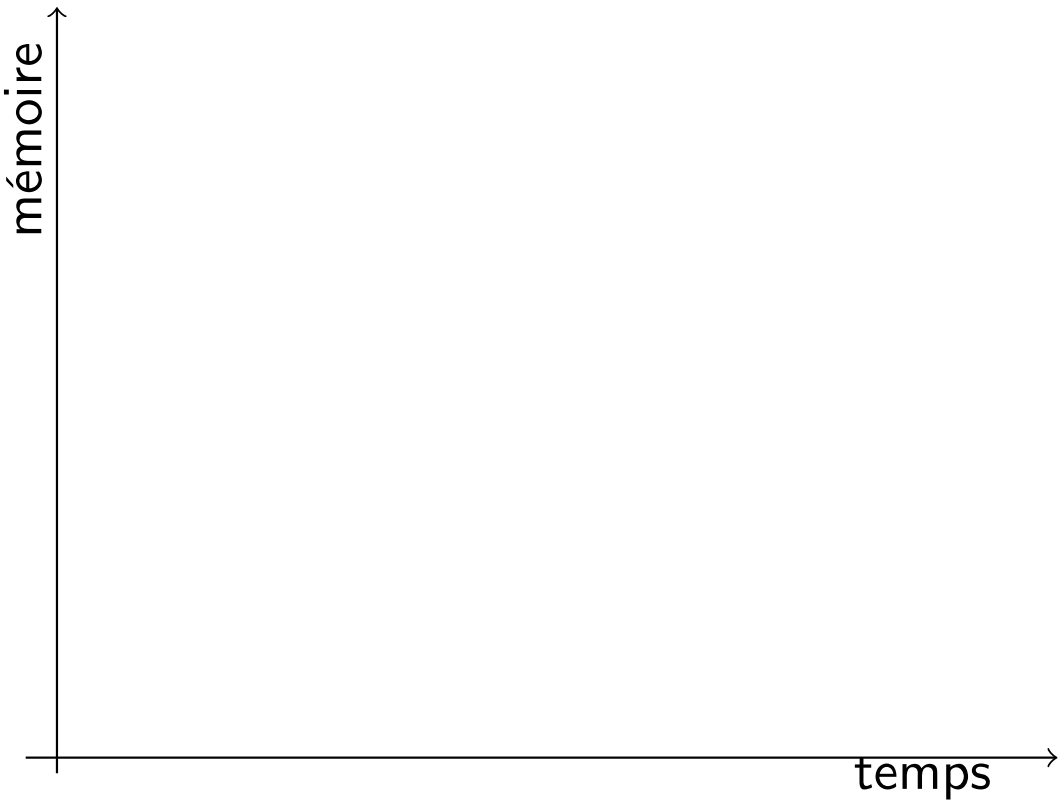
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



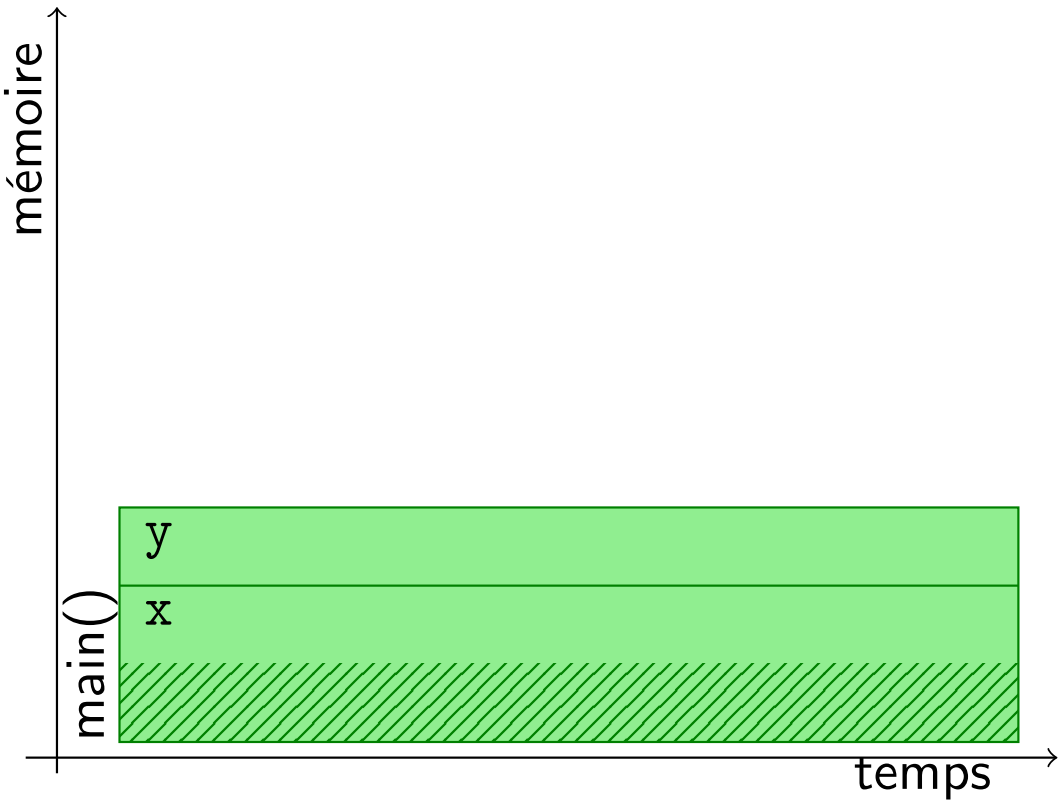
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



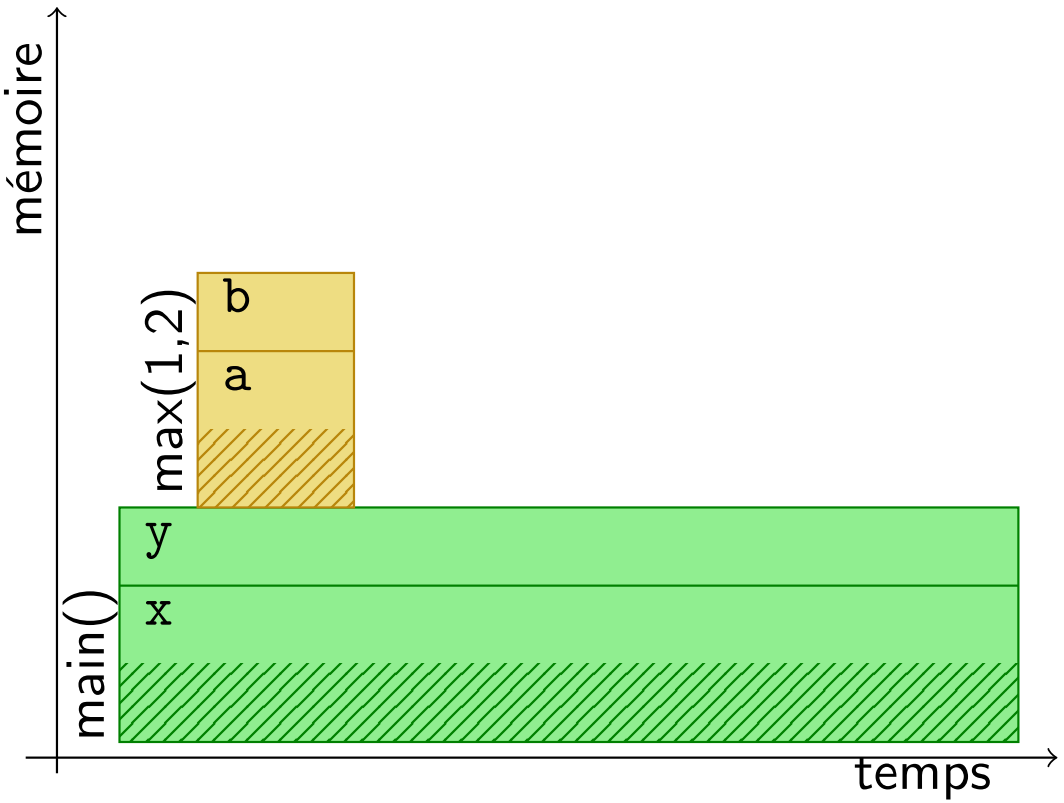
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



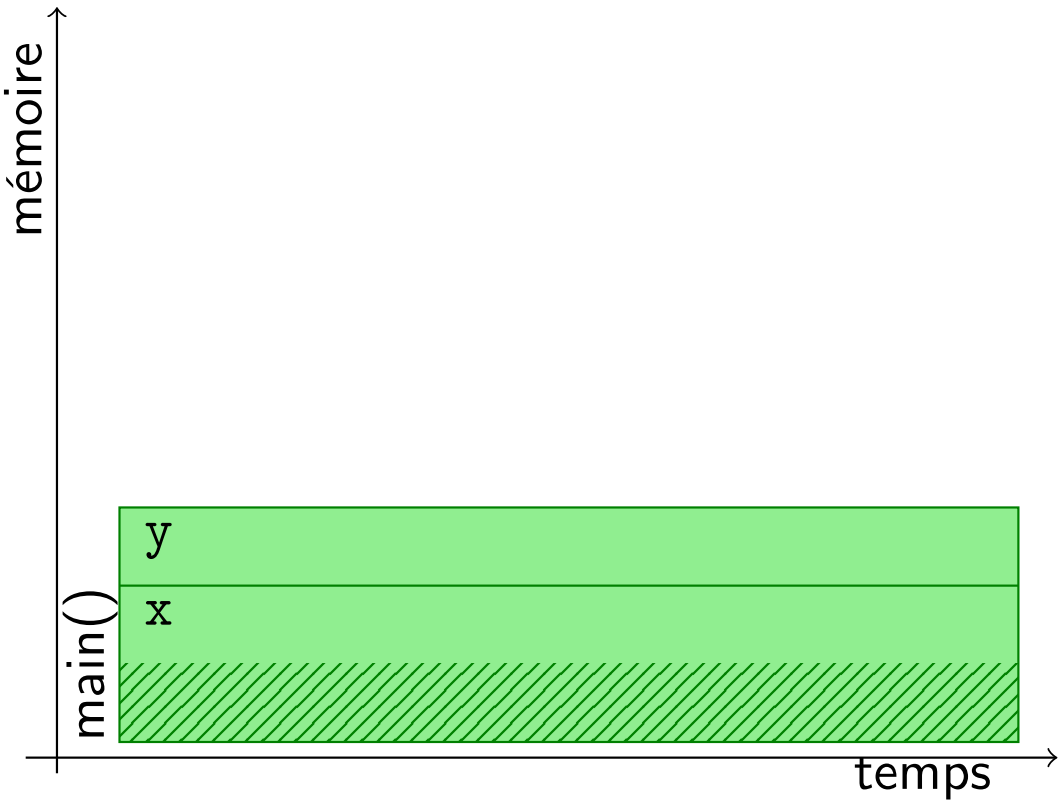
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



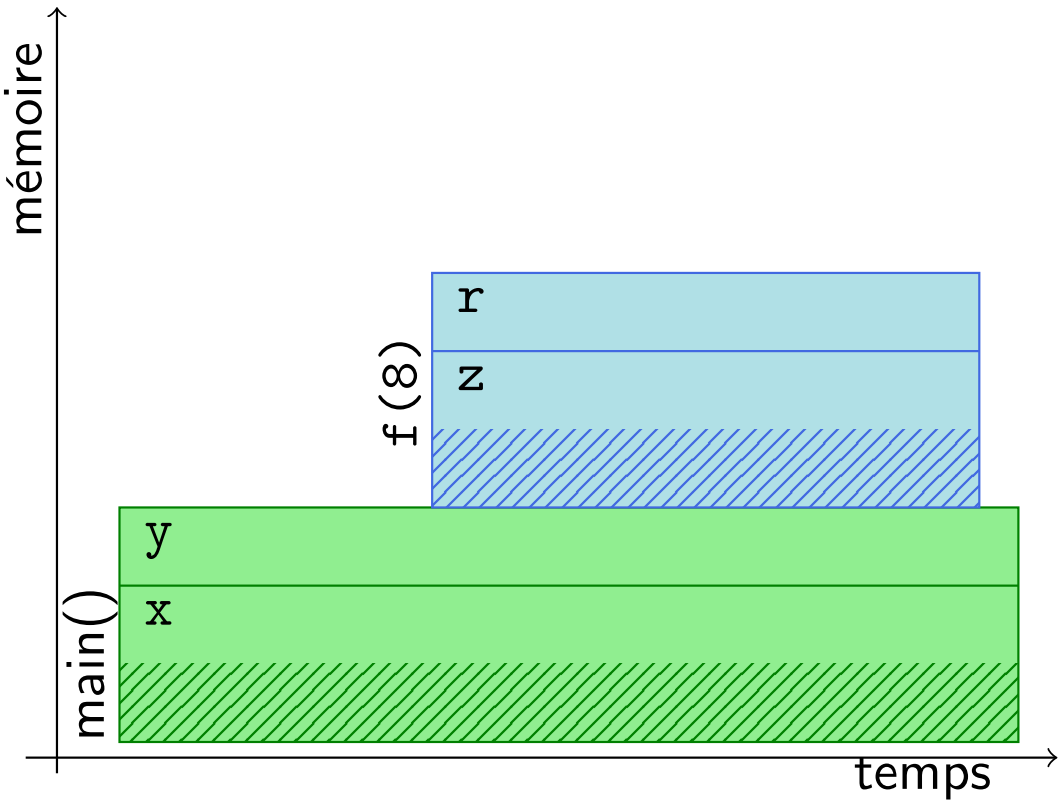
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



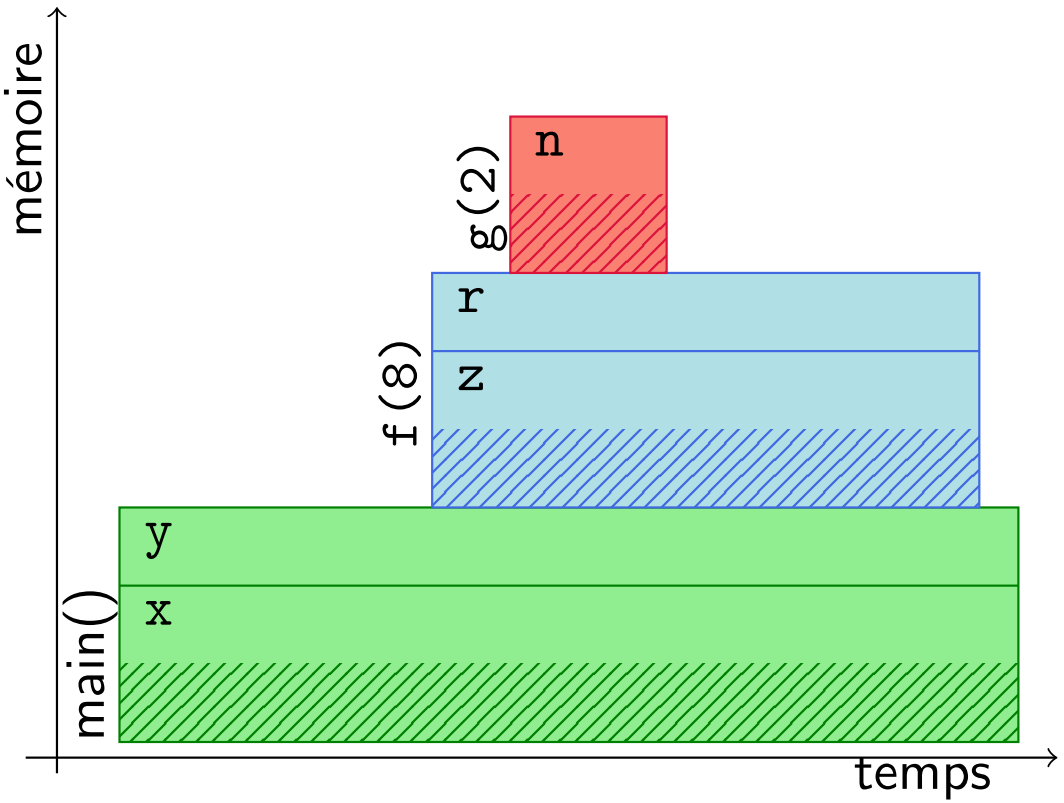
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



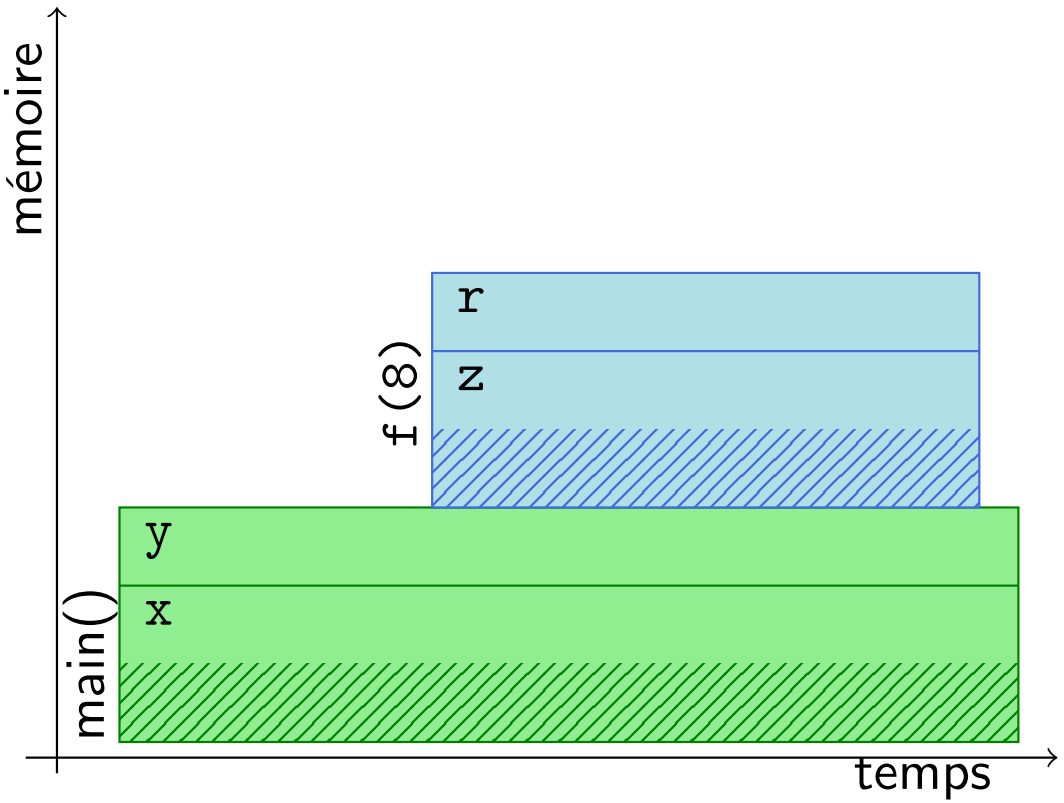
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



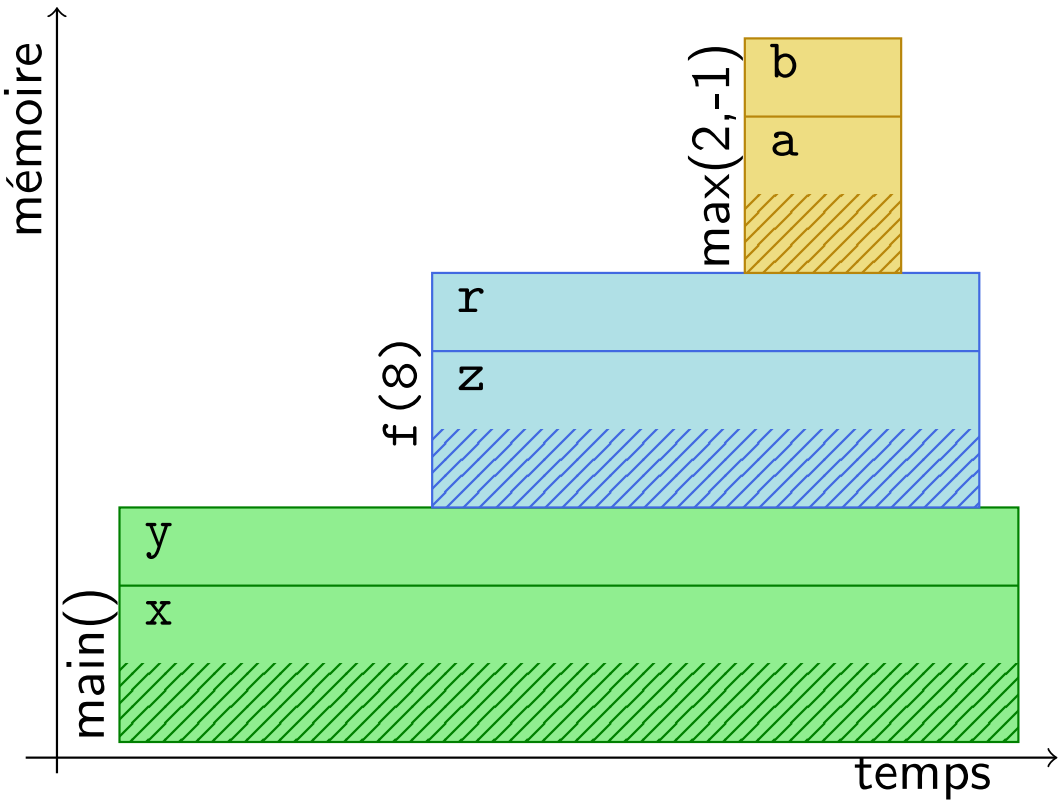
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



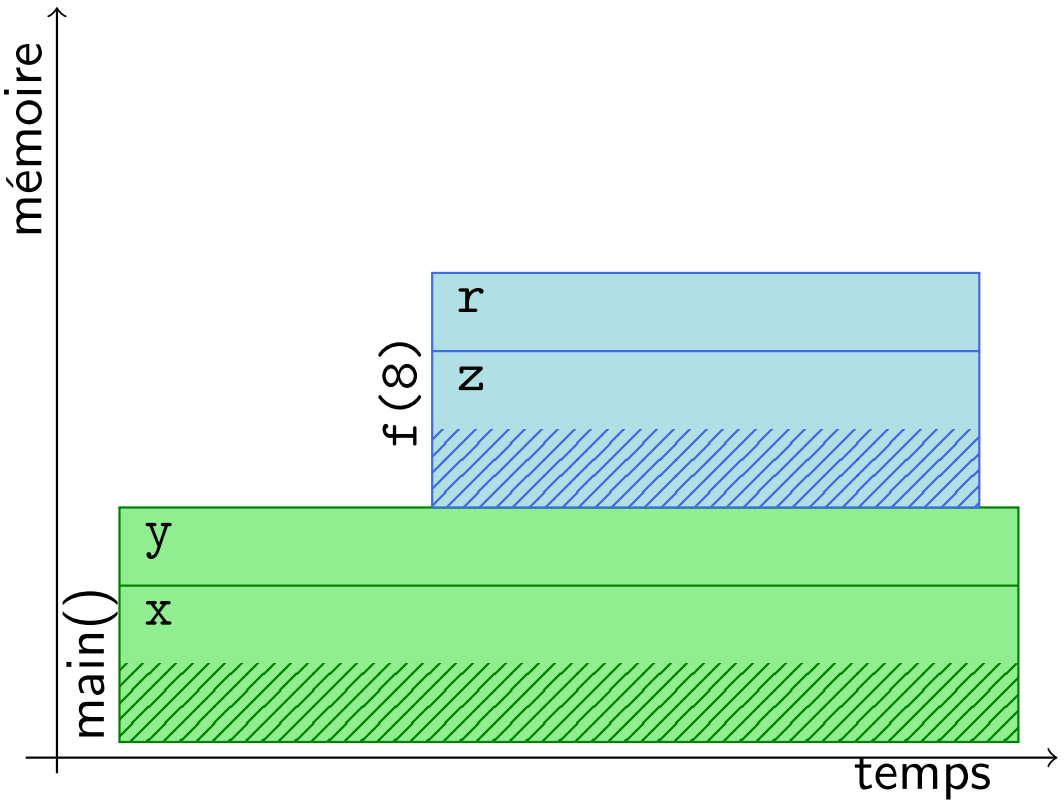
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



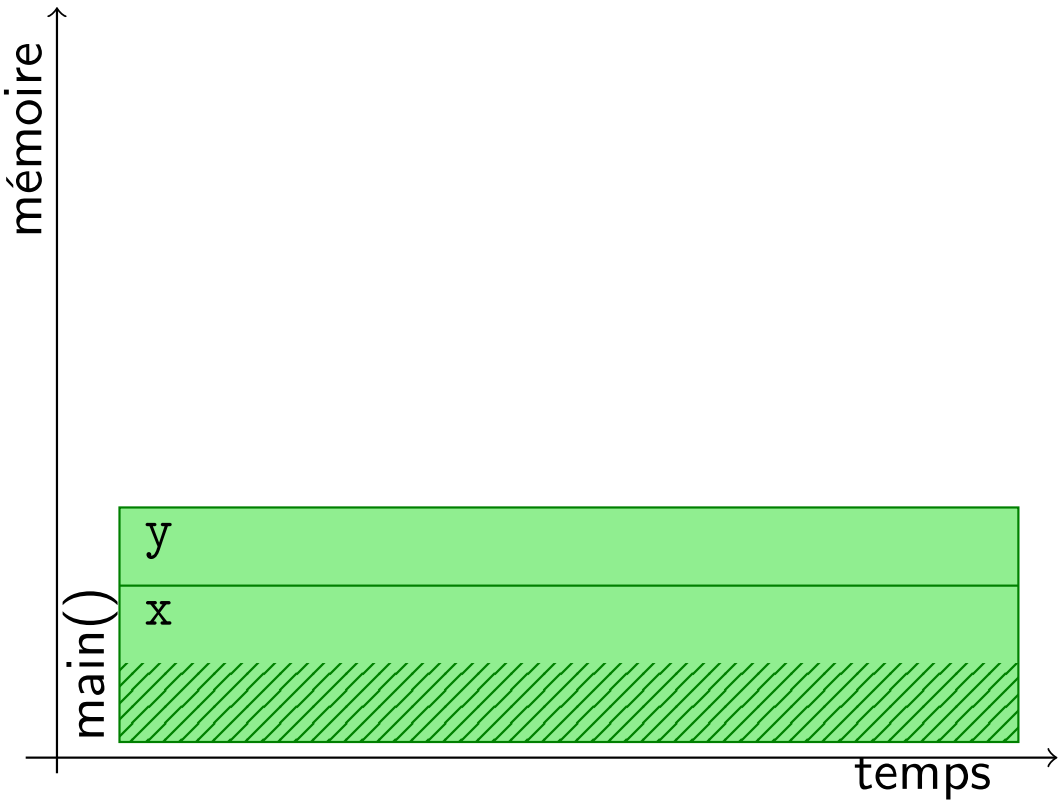
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



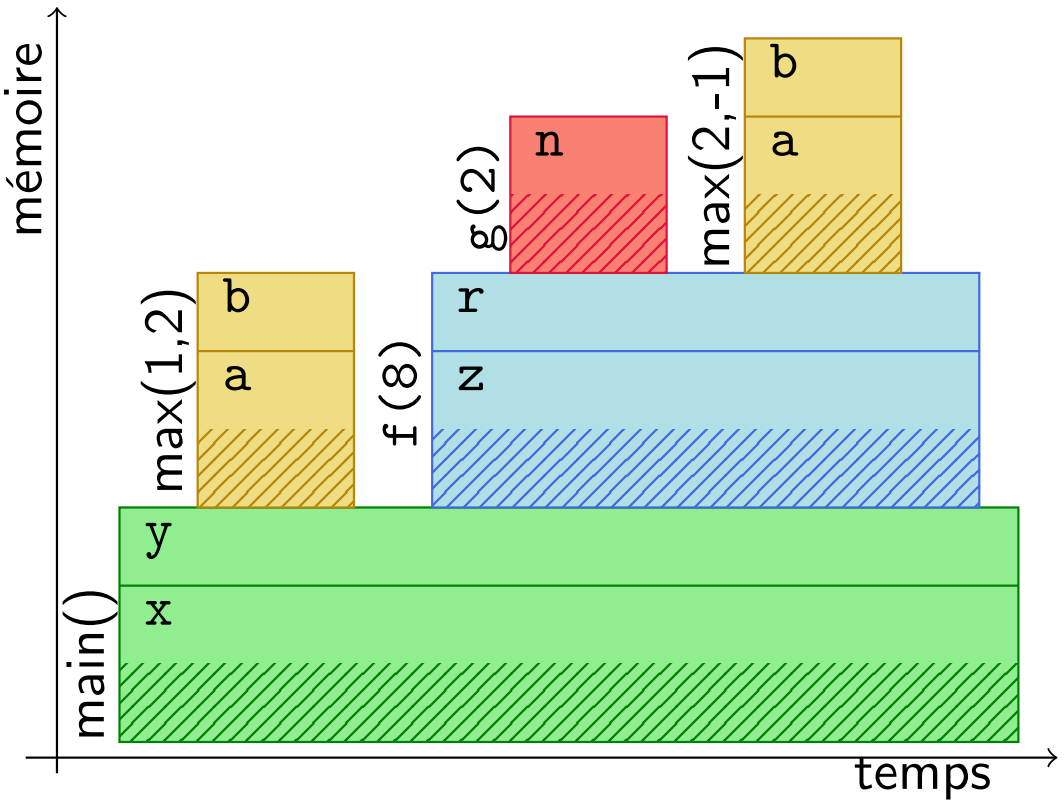
Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



Pile d'appel

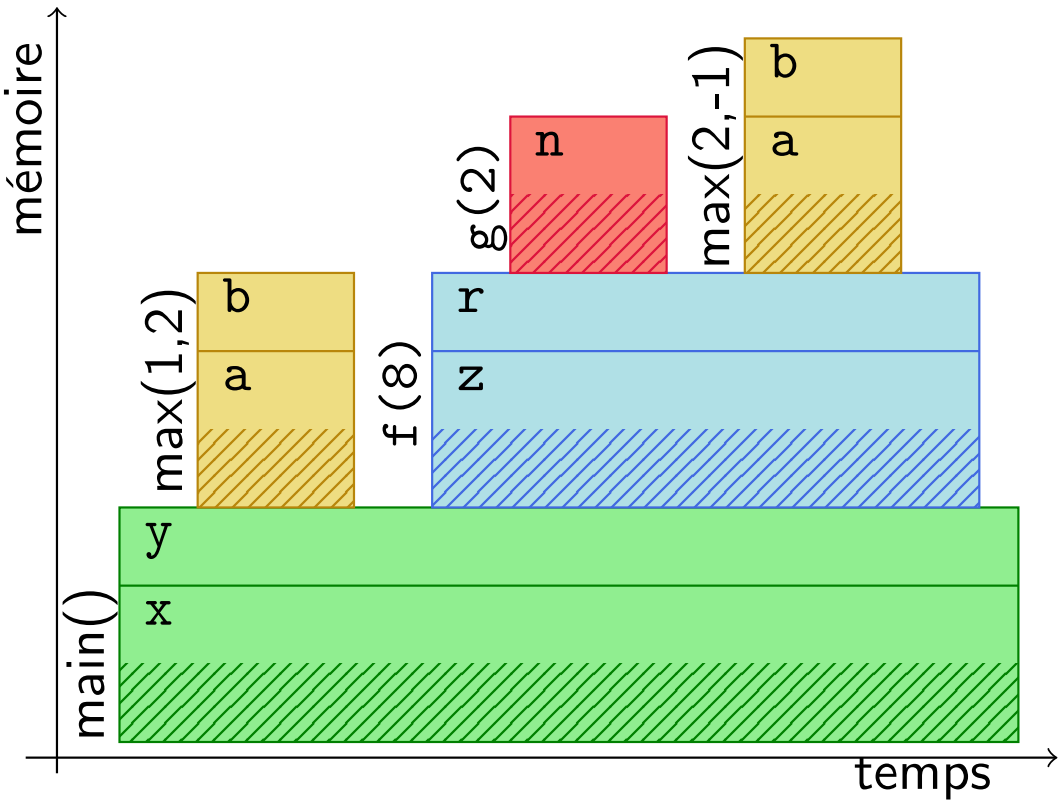
On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*



Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*

Peut-on avoir deux assiettes identiques dans la pile ? (La même fonction avec des contenus différents)



Factorielle récursive (teaser)

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Factorielle récursive (teaser)

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Factorielle récursive (teaser)

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

```
int factorielle(int n)
{
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

Utiliser les fonctions d'une bibliothèque (math.h)

Utilisation de la bibliothèque math.h

```
$ man math
```

Déclarer

```
#include <math.h>
```

Appeler

```
double x;

x = log(3.5);
```

Définir

```
$ gcc -lm -Wall prog.c -o prog.exe
```

Longue démo (menu)