

# CHAPITRE 1

## RAPPEL DU LANGAGE C

II1B

Université de La Manouba – ENSI – 2018/2019

## Plan

2

- ❑ Bref historique
- ❑ Caractéristiques du langage C
- ❑ Structure d'un programme C
- ❑ Compilation et exécution d'un programme C
- ❑ Adressage des variables
- ❑ Pointeurs
- ❑ Allocation dynamique de la mémoire
- ❑ Passage de paramètres à une fonction

## Bref historique

3

- 1972 : Dennis Ritchie développe le langage C dans les laboratoires Bell d'AT&T (dans le but d'écrire une version portable du système d'exploitation UNIX)
- 1978 : Dennis Ritchie et Brian Kernighan publient la définition classique du langage C dans un livre intitulé « *The C Programming Language* ».
- 1983 : l'ANSI (American National Standards Institute) met au point une version standard et portable du langage C baptisée ANSI-C.
- 1988 : Dennis Ritchie et Brian Kernighan publient la seconde édition du livre « *The C Programming Language* » respectant le standard ANSI-C.  
[http://net.pku.edu.cn/~course/cs101/2008/resource/The\\_C\\_Programming\\_Language.pdf](http://net.pku.edu.cn/~course/cs101/2008/resource/The_C_Programming_Language.pdf)

## Caractéristiques du langage C

4

### Avantages

- **Universel** : non orienté vers un domaine d'application particulier (ex: FORTRAN pour applications scientifiques et techniques, COBOL pour applications commerciales ou traitant de grandes quantités de données).
- **Compact** : basé sur un noyau de fonctions et d'opérateurs limité permettant la formulation d'expressions simples mais efficaces.
- **Moderne** : c'est un langage structuré, déclaratif; il offre des structures de contrôle et de déclaration comparables à celles des autres grands langages (FORTRAN, ALGOL68, PASCAL)

## Caractéristiques du langage C

5

### Avantages

- ❑ **Près de la machine** : étant développé dans le but de programmer le système d'exploitation UNIX, il offre des fonctions permettant un accès simple et direct aux fonctions internes de l'ordinateur (gestion de la mémoire).
- ❑ **Portable** : en respectant le standard ANSI-C, il est possible d'utiliser le même programme sur tout autre système simplement en le recompilant.
- ❑ **Extensible** : C ne se compose pas seulement des fonctions standard; le langage est animé par des bibliothèques de fonctions privées ou livrées par de nombreuses maisons de développement.

## Caractéristiques du langage C

6

### Limites

- ❑ **Compréhensibilité** : l'utilisation des expressions compactes entraîne le risque de produire du code incompréhensible d'où l'intérêt de bien commenter les programmes.
- ❑ **Portabilité** : le répertoire des fonctions standards étant limité, un programmeur peut donc être amené à utiliser des bibliothèques de fonctions prédéfinies non standards d'où le risque de perte en portabilité.
- ❑ **Discipline de programmation** : C n'impose pas un style de programmation strict, il offre beaucoup de liberté de codage et donc requiert beaucoup de responsabilité de la part du programmeur qui doit veiller à la propreté, la solidité et la compréhensibilité de son code.

## Structure d'un programme C

7

```
#include <stdio.h>
#define TAUX 2.063
```

Directives du préprocesseur

```
float convert(float euro);
```

Déclarations des fonctions

```
main()
{
    float somme;
    printf("Somme à convertir (en euros) : ");
    scanf("%f", &somme);
    printf("%f euros = %f dinars", somme, convert(somme));
}
```

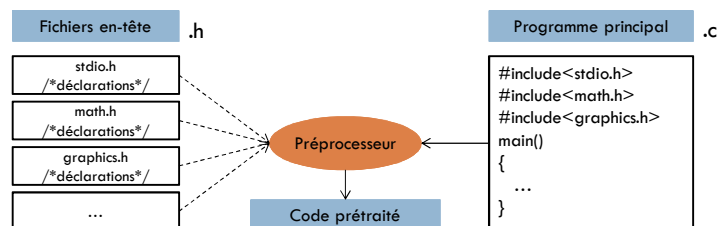
Programme principal

```
float convert(float euro)
{
    return(euro*TAUX);
}
```

Définitions des fonctions

## Compilation et exécution d'un programme C

8



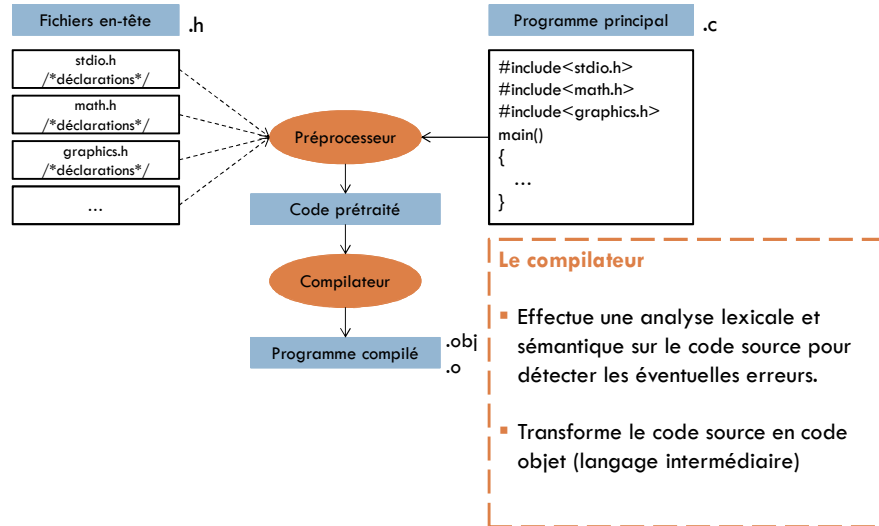
### Le préprocesseur

Assure une phase préliminaire de traitement textuel sur le code source.

- Supprime les commentaires (`/* ... */`)
- Procède à une évaluation des directives (commandes commençant par `#`)
  - Inclusion des fichiers d'en-tête (`#include`)
  - Remplacement des constantes (`#define`)

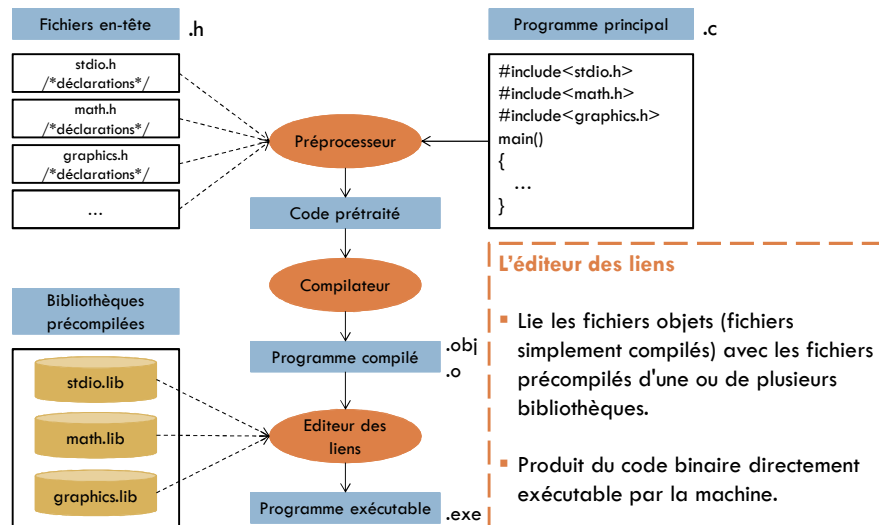
## Compilation et exécution d'un programme C

9



## Compilation et exécution d'un programme C

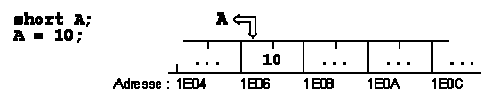
10



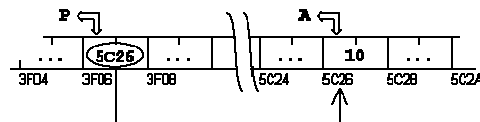
## Adressage des variables

11

- **Adressage direct** : Accès au contenu d'une variable par le nom de la variable.



- **Adressage indirect** : Accès au contenu d'une variable en passant par un pointeur qui contient l'adresse de cette variable.



## Variables simples

12

Code Source

<code>char</code>	<code>a = 1;</code>
<code>short int</code>	<code>b = 32000;</code>
<code>long int</code>	<code>c = 65536;</code>

Mémoire

Adresse	Nom	Contenu
0123456	a	1
0123457	b	32000
0123458		
0123459	c	65536
0123460		
0123461		
0123462		
0123463		?
0123464		?

## Tableaux

13

Code Source

<code>short int tab[4];</code>

Mémoire

Adresse	Nom	Contenu
0123456	tab	?
0123457		?
0123458		?
0123459		?
0123460		?
0123461		?
0123462		?
0123463		?
0123464		?

## Tableaux

14

Code Source

<code>short int tab[4];</code>

Mémoire

Adresse	Nom	Contenu
0123456	tab [0]	?
0123457		?
0123458		?
0123459		?
0123460		?
0123461		?
0123462		?
0123463		?
0123464		?

## Structures

15

Code Source

Struct record
{
int a;
short int b;
char c;
} rec;

Mémoire

Adresse	Nom	Contenu
0123456	rec	?
0123457		?
0123458		?
0123459		?
0123460		?
0123461		?
0123462		?
0123463		?
0123464		?

## Structures

16

Code Source

Struct record
{
int a;
short int b;
char c;
} rec;

Mémoire

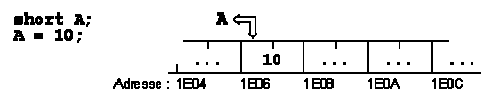
Adresse	Nom		Contenu
0123456	rec	a	?
0123457			
0123458			
0123459			?
0123460		b	
0123461			?
0123462		c	
0123463			?
0123464			?



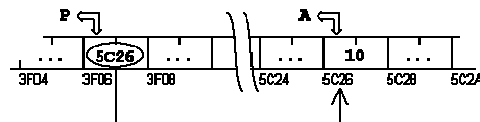
## Adressage des variables

17

- **Adressage direct** : Accès au contenu d'une variable par le nom de la variable.



- **Adressage indirect** : Accès au contenu d'une variable, en passant par un pointeur qui contient l'adresse de la variable.



## Pointeurs

18

Code Source

<code>int *ptr;</code>
<code>int b;</code>

Mémoire

Adresse	Nom	Contenu
0123456	ptr	?
0123457		
0123458		
0123459		
0123460	b	?
0123461		
0123462		
0123463		
0123464		0

## Pointeurs

19

Code Source

int *ptr;
int b;
ptr = &b;

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	b	?
0123461		
0123462		
0123463		
0123464		0

## Pointeurs

20

Code Source

int *ptr;
int b;
ptr = &b;
*ptr = 1;
b = 2;
b = b + *ptr;


Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	b	1
0123461		
0123462		
0123463		
0123464		0

## Pointeurs

21

Code Source

int *ptr;
int b;
ptr = &b;
*ptr = 1;
 b = 2;
b = b + *ptr;


Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	b	2
0123461		
0123462		
0123463		
0123464		0

## Pointeurs

22

Code Source

int *ptr;
int b;
ptr = &b;
*ptr = 1;
b = 2;
 b = b + *ptr;

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	b	4
0123461		
0123462		
0123463		
0123464		0

## Pointeurs sur tableaux

23

Code Source

<code>short int *ptr;</code>
<code>short int tab[4];</code>

Mémoire

Adresse	Nom	Contenu
0123456	ptr	?
0123457		
0123458		
0123459		
0123460	tab	?
0123461		?
0123462		?
0123463		?
0123464		?
0123465		?
0123466		?
0123467		?

## Pointeurs sur tableaux

24

Code Source

<code>short int *ptr;</code>
<code>short int tab[4];</code>
<code>ptr = tab;</code>
<code>*ptr = 3141;</code>
<code>ptr += 2;</code>
<code>*ptr = 512;</code>
<code>tab[3] = ptr - tab;</code>


Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	tab	[0] ?
0123461		[1] ?
0123462		[2] ?
0123463		[3] ?
0123464		[0] ?
0123465		[1] ?
0123466		[2] ?
0123467		[3] ?

## Pointeurs sur tableaux

25

Code Source

short int *ptr;
short int tab[4];
ptr = tab;
 *ptr = 3141;
ptr += 2;
*ptr = 512;
tab[3] = ptr - tab;


Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	tab [0]	3141
0123461		?
0123462		
0123463		
0123464		
0123465	[2]	?
0123466		
0123467		

## Pointeurs sur tableaux

26

Code Source

short int *ptr;
short int tab[4];
ptr = tab;
*ptr = 3141;
 ptr += 2;
*ptr = 512;
tab[3] = ptr - tab;


Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	tab [0]	3141
0123461		?
0123462		
0123463		
0123464		
0123465	[2]	?
0123466		
0123467		

## Pointeurs sur tableaux

27

Code Source

short int *ptr;
short int tab[4];
ptr = tab;
*ptr = 3141;
ptr += 2;
 *ptr = 512;
tab[3] = ptr - tab;


Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123464
0123457		
0123458		
0123459		
0123460	tab [0]	3141
0123461		?
0123462		
0123463		512
0123464		
0123465	[3]	?
0123466		
0123467		

## Pointeurs sur tableaux

28

Code Source

short int *ptr;
short int tab[4];
ptr = tab;
*ptr = 3141;
ptr += 2;
*ptr = 512;
 tab[3] = ptr - tab;

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123464
0123457		
0123458		
0123459		
0123460	tab [0]	3141
0123461		?
0123462		
0123463		512
0123464		
0123465	[3]	2
0123466		
0123467		

## Pointeurs sur structures

29

Code Source

struct record
{
short int a;
short int b;
} rec, *ptr;
ptr = &rec;
ptr->b = 16384;


Mémoire

Adresse	Nom	Contenu
0123456	ptr	?
0123457		
0123458		
0123459		
0123460	rec a	?
0123461		
0123462		b
0123463		
0123464		0

## Pointeurs sur structures

30

Code Source

struct record
{
short int a;
short int b;
} rec, *ptr;
 ptr = &rec;
ptr->b = 16384;

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	rec a	?
0123461		
0123462		b
0123463		
0123464		0

## Pointeurs sur structures

31

Code Source

struct record
{
short int a;
short int b;
} rec, *ptr;
ptr = &rec;
ptr->b = 16384;

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0123460
0123457		
0123458		
0123459		
0123460	rec a	?
0123461		
0123462		16384
0123463		0
0123464		0

## Allocation dynamique de la mémoire

32

- Opérateur : **sizeof**

```
sizeof(<type>)
sizeof(<variable>)
```
- Type : **size\_t** (~ unsigned long int)
- Exemple : **size\_t** len = **sizeof**(int);

Fichier <stdlib.h>	
void * <b>malloc</b> (size_t)	allocation d'un bloc
void * <b>calloc</b> (size_t, size_t)	allocation & initialisation d'un bloc
void * <b>realloc</b> (void *, size_t)	modification de la taille d'un bloc
void <b>free</b> (void *)	libération d'un bloc



## Allocation dynamique de la mémoire

33

Code Source

#include <stdlib.h>
{
void *ptr;
ptr = malloc(5);
ptr = realloc(ptr, sizeof(int));
free(ptr);
ptr = calloc(2, sizeof(short int));
free(ptr);
}

Mémoire

Adresse	Nom	Contenu
0123456	ptr	?
0123457		
0123458		
0123459		

## Allocation dynamique de la mémoire

34

Code Source

#include <stdlib.h>
{
void *ptr;
ptr = malloc(5);
ptr = realloc(ptr, sizeof(int));
free(ptr);
ptr = calloc(2, sizeof(short int));
free(ptr);
}

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0065536
0123457		
0123458		
0123459		

Adresse	Contenu
0065536	?
0065537	?
0065538	?
0065539	?
0065540	?
0065541	?

## Allocation dynamique de la mémoire

35

Code Source

#include <stdlib.h>
{
void *ptr;
ptr = malloc(5);
ptr = realloc(ptr, sizeof(int));
free(ptr);
ptr = calloc(2, sizeof(short int));
free(ptr);
}

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0065662
0123457		
0123458		
0123459		

Adresse	Cont.	Adresse	Cont.
0065536	?	0065662	?
0065537	?	0065663	?
0065538	?	0065664	?
0065539	?	0065665	?
0065540	?	0065666	?
0065541	?	0065667	?

## Allocation dynamique de la mémoire

36

Code Source

#include <stdlib.h>
{
void *ptr;
ptr = malloc(5);
ptr = realloc(ptr, sizeof(int));
free(ptr);
ptr = calloc(2, sizeof(short int));
free(ptr);
}

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0065662
0123457		
0123458		
0123459		

Adresse	Cont.
0065662	?
0065663	?
0065664	?
0065665	?
0065666	?
0065667	?

## Allocation dynamique de la mémoire

37

Code Source

#include <stdlib.h>
{
void *ptr;
ptr = malloc(5);
ptr = realloc(ptr, sizeof(int));
free(ptr);
ptr = calloc(2, sizeof(short int));
free(ptr);
}

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0065790
0123457		
0123458		
0123459		

Adresse	Contenu
0065790	0
0065791	
0065792	0
0065793	
0065794	?
0065795	?

## Allocation dynamique de la mémoire

38

Code Source

#include <stdlib.h>
{
void *ptr;
ptr = malloc(5);
ptr = realloc(ptr, sizeof(int));
free(ptr);
ptr = calloc(2, sizeof(short int));
free(ptr);
}

Mémoire

Adresse	Nom	Contenu
0123456	ptr	0065790
0123457		
0123458		
0123459		

Adresse	Contenu
0065790	0
0065791	0
0065792	0
0065793	0
0065794	?
0065795	?

## Appel de fonctions

39

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465		?
0123464		?
0123463		?
0123462		?
0123461		?
0123460		?
0123459		?
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

40

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464		?
0123463		?
0123462		?
0123461		?
0123460		?
0123459		?
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

41

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464	@Retour	<adresse>
0123463		
0123462		
0123461		
0123460	a	?
0123459	b	?
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

42

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464	@Retour	<adresse>
0123463		
0123462		
0123461		
0123460	a	6
0123459	b	?
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

43

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464	@Retour	<adresse>
0123463		
0123462		
0123461		
0123460	a	6
0123459	b	2
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

44

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464	@Retour	<adresse>
0123463		
0123462		
0123461		
0123460	a	8
0123459	b	2
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

45

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464		?
0123463		?
0123462		?
0123461		?
0123460		8
0123459		2
0123458		?
0123457		?
0123456		?

## passage de paramètres par valeur

46

Code Source

void f(char a)
{
char b = 2;
a += b;
}
void main()
{
char x = 6;
f(x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	9
0123464		?
0123463		?
0123462		?
0123461		?
0123460		8
0123459		2
0123458		?
0123457		?
0123456		?

## passage de paramètres par adresse

47

Code Source

void f(char *a)
{
char b = 2;
*a += b;
}
void main()
{
char x = 6;
f(&x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465		?
0123464		?
0123463		?
0123462		?
0123461		?
0123460		?
0123459		?
0123458		?
0123457		?
0123456		?

## passage de paramètres par adresse

48

Code Source

void f(char *a)
{
char b = 2;
*a += b;
}
void main()
{
char x = 6;
f(&x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465		?
0123464		?
0123463		?
0123462		?
0123461		?
0123460		?
0123459		?
0123458		?
0123457		?
0123456		?



## passage de paramètres par adresse

49

Code Source

void f(char *a)
{
char b = 2;
*a += b;
}
void main()
{
char x = 6;
f(&x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	6
0123464		?
0123463		?
0123462		?
0123461		?
0123460		?
0123459		?
0123458		?
0123457		?
0123456		?
0123455		

## passage de paramètres par adresse

50

Code Source

void f(char *a)
{
char b = 2;
*a += b;
}
void main()
{
char x = 6;
f(&x);
x += 3;
}

Mémoire



Adresse	Nom	Contenu
0123465	x	6
0123464	@retour	<adresse>
0123463		
0123462		
0123461		
0123460	a	?
0123459		
0123458		
0123457		
0123456	b	?
0123455		

## passage de paramètres par adresse

51

Code Source

Mémoire



	Adresse	Nom	Contenu
 void f(char *a)	0123465	x	6
{	0123464	@retour	<adresse>
char b = 2;	0123463		
*a += b;	0123462		
}	0123461		
void main()			
{			
char x = 6;	0123460	a	0123465
 f(&x);	0123459		
x += 3;	0123458		
}	0123457		
	0123456	b	?
	0123455		

## passage de paramètres par adresse

52

Code Source

Mémoire

	Adresse	Nom	Contenu
void f(char *a)	0123465	x	6
{	0123464	@retour	<adresse>
 char b = 2;	0123463		
*a += b;	0123462		
}	0123461		
void main()			
{			
char x = 6;	0123460	a	0123465
 f(&x);	0123459		
x += 3;	0123458		
}	0123457		
	0123456	b	2
	0123455		

## passage de paramètres par adresse

53

Code Source

Mémoire

	Adresse	Nom	Contenu
void f(char *a)	0123465	x	8
{	0123464	@retour	<adresse>
char b = 2;	0123463		
*a += b;	0123462		
}	0123461		
void main()	0123460	a	0123465
{	0123459		
char x = 6;	0123458		
f(&x);	0123457		
x += 3;	0123456	b	2
}	0123455		

## passage de paramètres par adresse

54

Code Source

Mémoire

	Adresse	Nom	Contenu
void f(char *a)	0123465	x	8
{	0123464		?
char b = 2;	0123463		?
*a += b;	0123462		?
}	0123461		?
void main()	0123460		?
{	0123459		?
char x = 6;	0123458		?
f(&x);	0123457		?
x += 3;	0123456		2
}	0123455		?

## passage de paramètres par adresse

55

Code Source

void f(char *a)
{
char b = 2;
*a += b;
}
void main()
{
char x = 6;
f(&x);
x += 3;
}

Mémoire

Adresse	Nom	Contenu
0123465	x	11
0123464		?
0123463		?
0123462		?
0123461		?
0123460		?
0123459		?
0123458		?
0123457		?
0123456		2
0123455		?