Programmation en C
Fonctions et passage de paramètres

Alain CROUZIL
alain.crouzil@irit.fr

Département d'Informatique (DdI)

Institut de Recherche en Informatique de Toulouse (IRIT)
Équipe Computational Imaging and Vision (MINDS)

Faculté Sciences et Ingénierie (FSI)
Université Toulouse III – Paul Sabatier (UPS)

Licence Informatique – Licence MIASHS
2019-2020

D[i] Département Informatique

UNIVERSITE TOULOUSE III PAUL SABATIER
Faculté Sciences et Ingénierie

IRIT
Institut de Recherche en Informatique de Toulouse

Sommaire

Sommaire

Introduction

Fonction en langage C

La fonction :

- est la seule sorte de sous-programme existant en C ;
- joue un rôle très général incluant celui des fonctions et des procédures d'autres langages.

Définition d'une fonction

```
type_retourné nom_fonction(déclaration_paramètres)
{
  corps_de_la_fonction
}
```

Sommaire

Valeur de retour

Instruction `return`

Dans une fonction, l'instruction :

```
return expression;
```

permet :

- de préciser la valeur de retour de la fonction (la valeur de *expression*) ;
- d'interrompre l'exécution de la fonction, c'est-à-dire de « redonner la main » à l'appelant.

Propriétés de la valeur de retour

- Une fonction peut retourner une valeur de n'importe quel type sauf de type tableau.
- Si une fonction ne retourne aucune valeur, son *type_retourné* est **void**.
- Si l'on omet le *type_retourné*, le compilateur considère qu'il s'agit du type **int**.
- La valeur de retour d'une fonction peut être ignorée par l'appelant.

Sommaire

Paramètres

Déclaration des paramètres

$$type_1\ ident_1, type_2\ ident_2, \ldots, type_n\ ident_n$$

Si une fonction n'a aucun paramètre, cette déclaration se réduit à `void`.

Mode de passage

Le passage des paramètres ne se fait que par valeur.

Transmission d'informations à l'appelant

Une fonction a trois possibilités directes pour transmettre des informations à l'appelant :

- par sa valeur de retour ;
- par l'intermédiaire de variables globales ;
- par ses paramètres, en utilisant les pointeurs.

Sommaire

Déclaration d'une fonction

Deux formes

$$type\_retourné\ nom\_fonction(type_1\ ident_1, \ldots, type_n\ ident_n);$$

ou bien

$$type\_retourné\ nom\_fonction(type_1, \ldots, type_n);$$

Déclaration, définition et utilisation

- Si une fonction est utilisée sans avoir été ni déclarée, ni définie au préalable, le compilateur considère qu'elle retourne une valeur de type **int**.
- Dans tous les cas, pour que l'éditeur de liens puisse produire le fichier exécutable, la définition d'une fonction doit être effectuée quelque part, soit dans le même fichier, soit dans un autre fichier (compilation séparée).

Sommaire

Utilisation des fonctions (1/3)

Quand peut-on utiliser une fonction ?

Une fonction peut être appelée si elle a été préalablement *définie* ou bien *déclarée*.

Utilisation après définition

```
/* Définition de f1 */
... f1(...)
{
   ...
   ...
}

/* Définition de f2 */
... f2(...)
{
   ...
   ...f1(...); /* Appel de f1 */
   ...
}
```

Utilisation des fonctions (2/3)

Utilisation après déclaration globale

```
/* Déclaration globale de f1 */
... f1(...);
/* Elle peut être utilisée partout dans la suite */

/* Définition de f2 */
... f2(...)
{
   ...
   ...f1(...); /* Appel de f1 */
   ...
}

/* Définition de f1 */
... f1(...)
{
   ...
   ...
}
```

Utilisation des fonctions (3/3)

Utilisation après déclaration locale

```
/* Définition de f2 */
... f2(...)
{
   ...
   /* Déclaration locale de f1 */
   ... f1(...);
   /* Elle ne peut être utilisée que dans f2 */
   ...
   ...f1(...); /* Appel de f1 */
   ...
}

/* Définition de f1 */
... f1(...)
{
   ...
   ...
}
```

Sommaire

Utilisation des pointeurs (1/10)

Passage par valeur

Lors de l'appel d'une fonction, la valeur du paramètre effectif est copiée dans le paramètre formel.

Conséquence

Pour qu'une fonction appelée puisse modifier la valeur d'une variable de l'appelant, il faut que l'appelant lui passe l'adresse de cette variable.

Démo

Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```

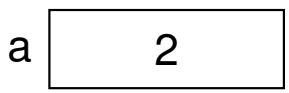
Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```



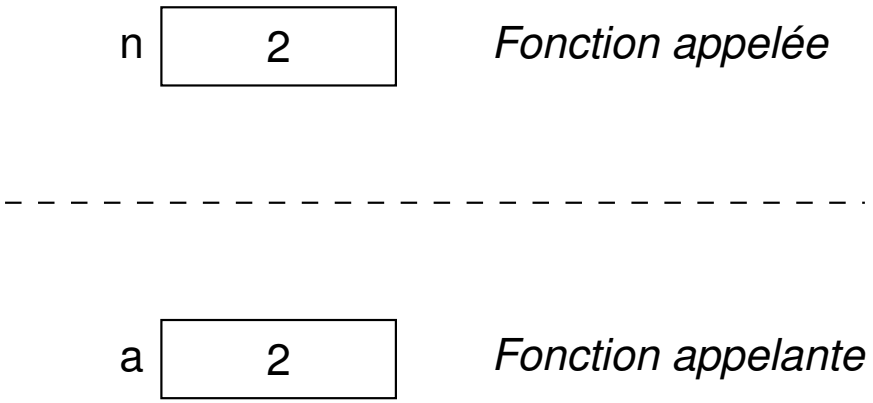
Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```



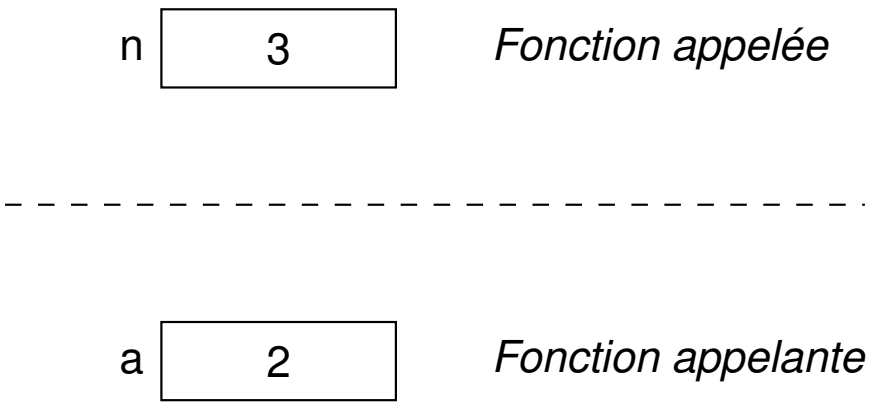
Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```



Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```

Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```

a | 2

Affichage de `a=2`

Utilisation des pointeurs (2/10)

Exemple : fonction inutile

```
void incr(int n)
{
  n = n + 1;
}

int main(void)
{
  int a=2;
  incr(a);
  printf("a=%d\n",a);

  return 0;
}
```

Utilisation des pointeurs (3/10)

☼ Principe

Quand une fonction appelante veut qu'une fonction appelée modifie un objet :

- l'**appelante** doit passer l'adresse de l'objet ;
- le paramètre formel de l'**appelée** doit être un pointeur vers l'objet ;
- l'**appelée** doit manipuler l'objet pointé.

Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}

int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);

  return 0;
}
```

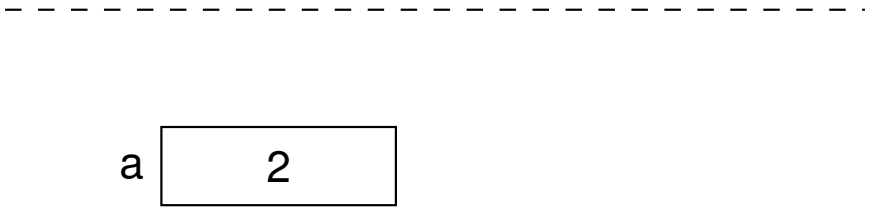
Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}


int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);


  return 0;
}
```



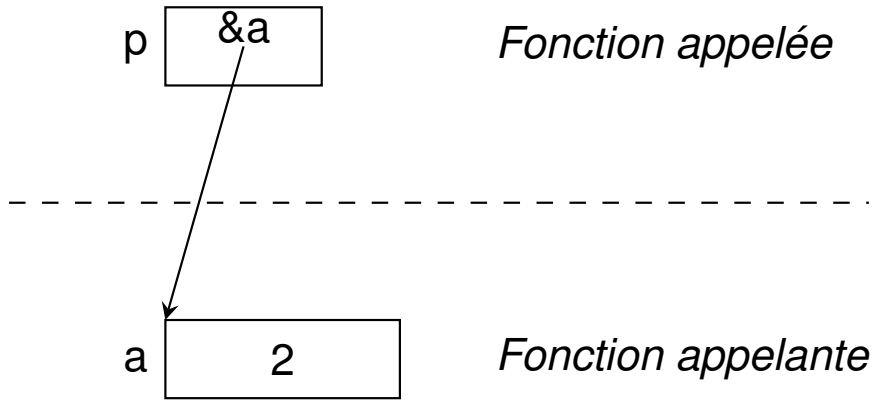
Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}

int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);

  return 0;
}
```



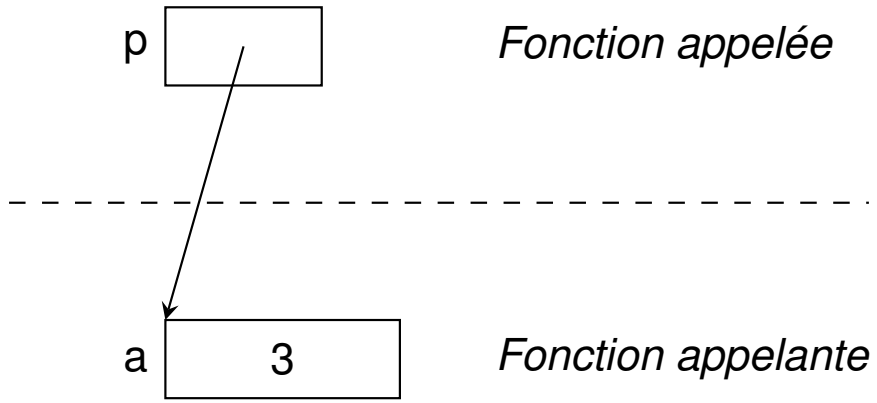
Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}

int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);

  return 0;
}
```



Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}

int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);

  return 0;
}
```

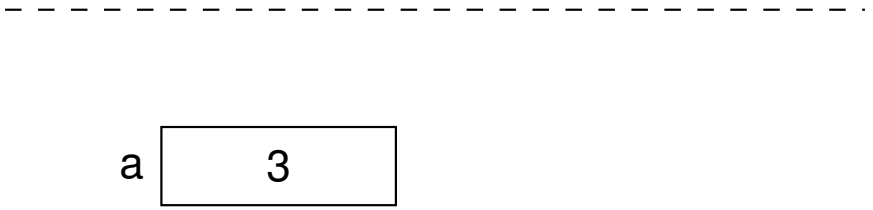
Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}

int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);

  return 0;
}
```



Affichage de `a=3`

Utilisation des pointeurs (4/10)

Exemple : fonction d'incrémentation

```
void incr(int *p)
{
  *p = *p + 1;
}

int main(void)
{
  int a=2;
  incr(&a);
  printf("a=%d\n",a);

  return 0;
}
```

Utilisation des pointeurs (5/10)

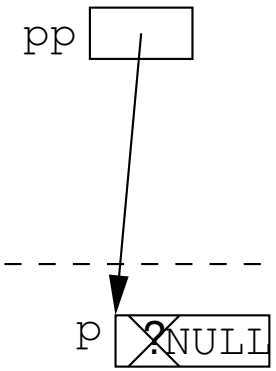
Exemple : fonction modifiant un pointeur

Appelée

```
void init(int **pp)
{
  *pp = NULL;
}
```

Appelante

```
int *p;
init(&p);
```



Utilisation des pointeurs (6/10)

Exemple : utilisation de la valeur de retour

Appelée

```
double carre1(double x)
{
  double r;
  r = x * x;
  return r;
}
```

x 8.0 r ~~?~~ 64.0

Appelante

```
double a=8.0,b;
b=carre1(a);
```

a 8.0 b ~~?~~ 64.0

Utilisation des pointeurs (7/10)

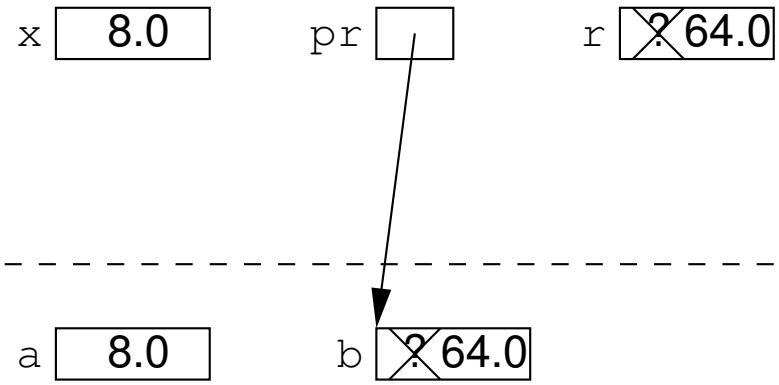
Exemple : utilisation des paramètres

Appelée

```
void carre2(double x, double *pr)
{
  double r;
  r = x * x;
  *pr = r;
}
```

Appelante

```
double a=8.0,b;
carre2(a,&b);
```



À vos boitiers !

10

À vos boitiers !

11

À vos boitiers !

12

À vos boitiers !

13

Utilisation des pointeurs (8/10)

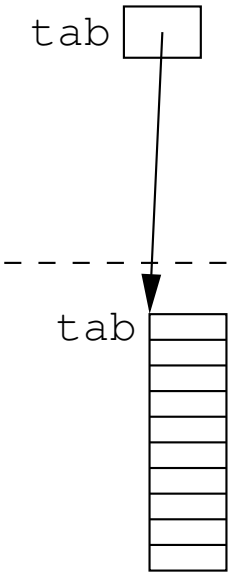
Passage d'un tableau en paramètre

Appelé

```
void fct(int tab[])
{
...
}
```

Appelant

```
int tab[10];
fct(tab);
```



Un paramètre formel de type tableau est en fait un pointeur (vers le premier élément du tableau effectivement passé à la fonction). Dans l'exemple précédent, on aurait pu aussi écrire : `void fct(int *tab)`

Utilisation des pointeurs (9/10)

Exemple pour un tableau à une dimension

```c
/* Affichage des éléments d'un
 * tableau de int
 */
void AffTab1(int Tab[], int NbElt)
{
  for (int i=0; i<NbElt; i++)
    printf("%d ",Tab[i]);
  printf("\n");
}

/* Incrémentation des éléments d'un
 * tableau de int
 */
void IncrTab1(int Tab[], int NbElt)
{
  for (int i=0; i<NbElt; i++)
    Tab[i]=Tab[i]+1;
}
```

```c
/* Exemple d'appel
 */
int main(void)
{
  int Tab1[5]={1,2,3,4,5};

  IncrTab1(Tab1,5);
  AffTab1(Tab1,5);

  return 0;
}
```

Utilisation des pointeurs (10/10)

Exemple pour un tableau à deux dimensions

Dans la déclaration du paramètre formel, le nombre de colonnes doit être précisé.

```
/* Affichage des éléments d'une matrice de int */
void AffTab2(int Tab[][3], int NbLig, int NbCol)
{
  for (int i=0; i<NbLig; i++)
  {
    for (int j=0; j<NbCol; j++)
      printf("%d ",Tab[i][j]);
    printf("\n");
  }
}

int main(void)
{
  int Tab2[2][3]={{1,2,3},{4,5,6}};

  AffTab2(Tab2,2,3);

  return 0;
}
```