Les fonctions et les procédures en C

Séance 6

de l'UE « introduction à la programmation »

Bruno Bouzy

bruno.bouzy@parisdescartes.fr

Fonctions et procédures

- Fonction, déclaration, définition, utilisation
- Type de retour, paramètres
- Procédure
- Paramètres en entrée ou en sortie
- Récursivité et itérations
- Macros
- Exemple complet

Fonctions

- Déclaration:

```
int mafonction(int);
```

- Définition:

```
int mafonction(int x) {
   int a; ...
   return(a);
}
```

- Utilisation:

```
int x = 5; int y = mafonction(x);
```

Fonctions

- Type de retour:

```
int mafonction(int);
```

- Type du paramètre:

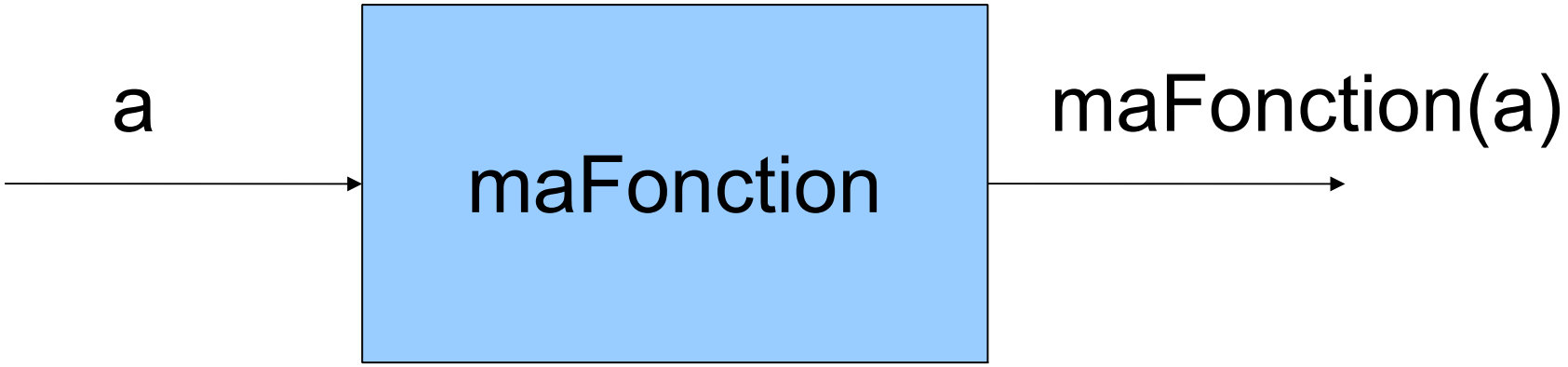
```
int mafonction(int);
```

- Nom:

```
int mafonction(int);
```

- Signature ou en-tête d'une fonction.

Boites et flèches...



Exemple 1

- Calcul du carré d'un nombre

```
int carre(int a) {          // definition
  return (a*a);             // retour
}

int main()
{
  int n, x=0;
  printf("n ? "); scanf("%d", &n);
  x = carre(n);                    // appel
  printf("carre = %d\n", x);
  return 0;
}
```

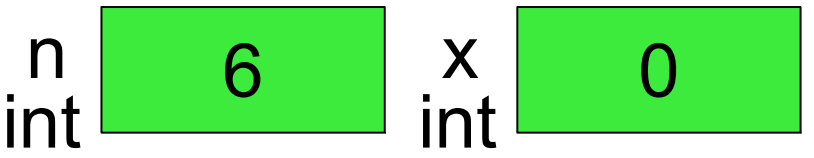
Exemple 1

- Exécution: ça marche!

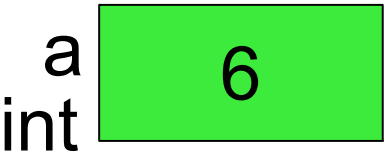
```
ProgC > ./a.out
n ? 6
carre = 36
ProgC >
```

Exemple 1

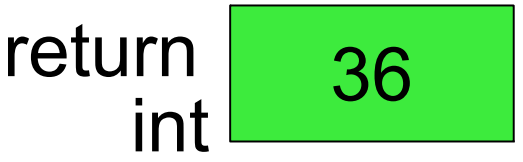
- Comment `carre` marche ?
  - Après le `scanf`:



  - Appel de la fonction:



  - Exécution du `return`



  - Retour dans l'appelant

Types de retour

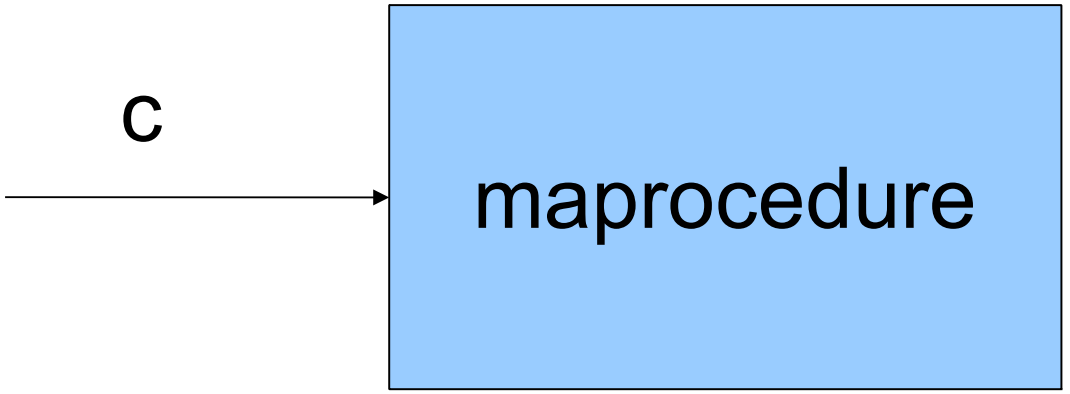
- Les types pré-définis pour les variables
  - `int, float, double, char`, etc.
  - `int *, float *, char *`, etc
- Le type vide:
  - `void`

Procédures (1/3)

- Le type vide:
  - `void`
- Une procédure:

```
void maprocedure(char);    // declaration
void maprocedure(char) {   // definition
   ...
}
char c;
maprocedure(c);                 // appel
```

Procédures (2/3)



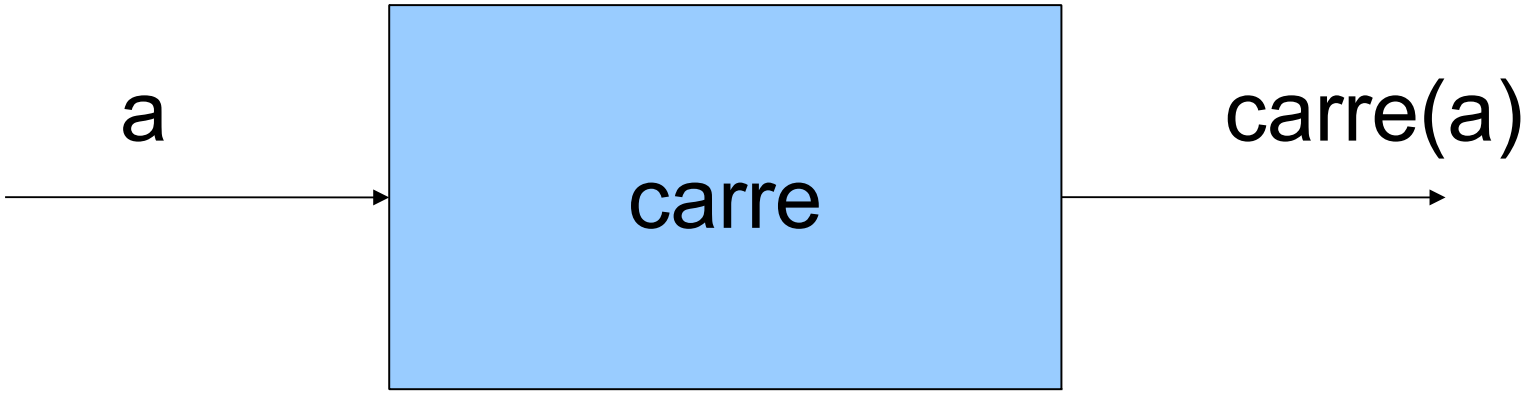
Procédures (3/3)

- Pas de paramètre de retour
  - Type `void`
- Structuration du programme
  - Un traitement donné est écrit une fois et une seule dans le programme (comme pour les fonctions)
- Possibilité d'avoir des paramètres en sortie
  - Une procédure peut donner un résultat

Paramètres en entrée ou sortie (1/3)

- Paramètre en entrée
  - La fonction ou la procédure en a besoin pour fonctionner
- Paramètre en sortie
  - La fonction ou la procédure a pour objectif de lui donner une valeur.
  - Si le type du retour n'est pas `void`, une fonction C possède un paramètre de sortie obligé: le paramètre de retour.

Paramètres en entrée ou sortie (2/3)



Paramètres en entrée ou sortie (3/3)

- En langage C, il n'existe pas d'autre paramètre de sortie que le paramètre de retour :-(
- Passage de paramètre par valeur
  - On passe la valeur du paramètre
  - Implique que le paramètre est en entrée
- Passage de paramètre par adresse
  - On passe l'adresse du paramètre
  - Cela permet d'avoir un paramètre en sortie :-)

Exemple 2

- Carré avec passage de paramètres par **adresse**

```
void carre2(int a, int * b) {
  *b = a*a;
}
int main()
{
  int n, x=0;
  printf("n ? ");
  scanf("%d", &n);
  carre2(n, &x);
  printf("carre2 = %d\n", x);
  return 0;
}
```

Exemple 2

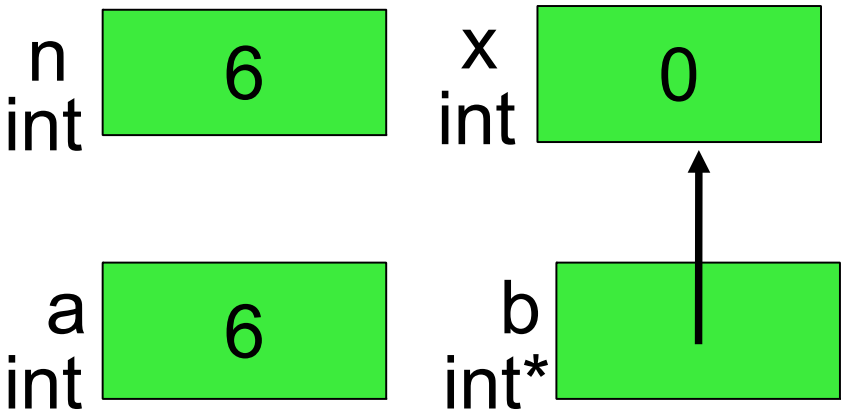
- Exécution: ça marche!

```
ProgC > ./a.out
n ? 6
carre2 = 36
ProgC >
```

Exemple 2

- Comment `carre2` marche ?
  - Après le `scanf`:
  - Appel de la fonction:

```
a = n; b = &x;
```



  - Exécution: `*b = a*a;`

  - Retour dans l'appelant

Exemple 2

Exemple 3

- Carré avec passage de paramètres par **valeur**

```
void carre3(int a, int b) {
  b = a*a;
}
int main()
{
  int n, x=0;
  printf("n ? ");
  scanf("%d", &n);
  carre3(n, x);
  printf("carre3 = %d\n", x);
  return 0;
}
```

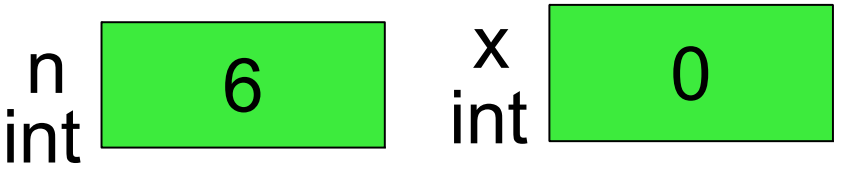
Exemple 3

- Exécution: ça marche pas :-(

```
ProgC > ./a.out
n ? 6
carre3 = 0
ProgC >
```

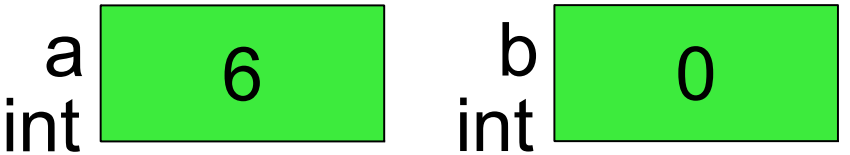
Exemple 3

- Comment `carre3` **marche pas** ?
  - Après le `scanf`:

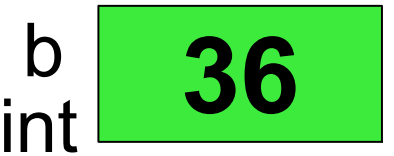


  - Appel de la fonction:

```
a = n; b = x;
```



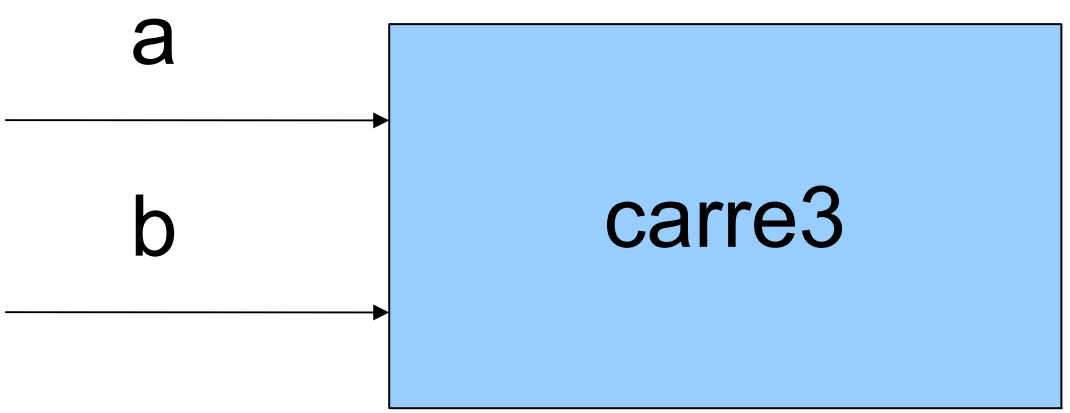
  - Exécution `b = a*a;`



  - Retour dans l'appelant

- Explication:
  - `b` a été modifié correctement, pas `x`.

Exemple 3



Recursivité

- Une fonction qui s'appelle soi-même

```
int factorielle(int n)
{
  if (n==0) return 1;
  return (n*factorielle(n-1));
}
```

```
int n = 4;
printf("n! = %d\n", factorielle(n));
```

Recursivité

- Factorielle sans récursivité, avec boucle:

```
int factorielle2(int n)
{
  if (n==0) return 1;
  int r=1;
  int i;
  for (i=1; i<=n; i++) r *= i;
  return r;
}

int n = 4;
printf("n! = %d\n", factorielle2(n));
```

Macros

- Definition

```
#define MA_MACRO(X, Y) X+19*Y
```

- Utilisation

```
int i = MA_MACRO(4, 13);
```

- `gcc` remplace `MA_MACRO(a, b)` par `a + 19*b` partout dans le source.

Macros

- Avantage:
  - rapidité d'exécution
  - lisibilité du source (comme pour les fonctions)

- Inconvénients:
  - effet de bords indésirés
  - expansion du code exécutable

Exemple complet (1/4)

- Ecrire une fonction:

```
int deIntervalleANombre(int a, int b)
```

  - demandant à l'U un nombre entier dans [a, b],
  - retournant ce nombre,
  - (la demande est renouvelée tant que l'U n'a pas tapé une valeur dans [a, b])

Exemple complet (2/4)

```
int deIntervalleANombre(int a, int b)
{
  int x;

  do {
    printf("x ? (%d<=x<=%d) ", a, b);

    scanf("%d", &x);

  } while (x<a || x>b);

  return x;
}
```

Exemple complet (3/4)

- Ecrire un programme principal `main`:
  - demandant un nombre N dans [0, 12]
  - avec la fonction `deIntervalleANombre`
  - affichant la valeur de N !

Exemple complet (4/4)

```
#include <stdio.h>

// ici les definitions des 2 fonctions

int main() {
  int n;

  n = deIntervalleANombre(0,12);

  printf("n = %d\n", n);

  printf("n! = %d\n", factorielle(n));

  return 0;
}
```

Résumé de la séance 6

- Fonction, déclaration, définition, utilisation
- Type de retour, paramètres
- Procédure
- Paramètres en entrée ou en sortie
- Passage de paramètres par adresse
- Récursivité et itérations
- Macros
- Exemple