# INFORMATIQUE 1

## VIII. FONCTIONS ET PROCÉDURES

```c
BinTree* createTree(int val){
    BinTree* pT = malloc(sizeof(BinTree));
    if(pT == NULL){
        exit(1);
    }
    pT->value  = val;
    pT->lChild = NULL;
    pT->rChild = NULL;
    return pT;
}

// Ex2-1d
int isEmpty(BinTree* pT){
    return (pT == NULL);
}

// Ex2-1e
int isLeaf(BinTree* pT){
    int result = 0;
    if(pT != NULL){
        result = (pT->lChild == NULL) && (pT->lChild == NULL);
    }
    return result;
}
```

Eva Ansermin & Romuald Grignon

CY TECH

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| | | |



Eva ANSERMIN & Romuald GRIGNON

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | | |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 8 | |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 8 | 2 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 6 | 2 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 6 | 3 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 6 | 3 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 4 | 3 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 4 | 4 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 4 | 4 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 2 | 4 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 2 | 5 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 2 | 5 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 0 | 5 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 0 | 6 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | 0 | 6 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -2 | 6 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -2 | 7 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -2 | 7 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -4 | 7 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -4 | 8 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -4 | 8 |

# Rappel cours précédent

```
PROGRAMME Oracle
VARIABLES
      a,b,i : entier
DÉBUT
      a <- 2
      b <- 8
      POUR i DE 2 A 8 FAIRE
         b <- b-a
      FIN POUR
      ECRIRE(b)
FIN
```

| a | b | i |
|---|---|---|
| 2 | -4 | 8 |

-4

# Un exemple de code pas très pratique…

```
PROGRAMME sondage
VARIABLES
      rep1, rep2, rep3 : entier
DÉBUT
       REPETER
           ECRIRE(“Vous préférez: 1: les chats 2: les chiens?”)
           LIRE(rep1)
      TANT QUE(rep1≠1 ET rep1≠2)
      REPETER
           ECRIRE(“Vous préférez: 1: les pâtes 2: le riz?”)
           LIRE(rep2)
      TANT QUE(rep2≠1 ET rep2≠2)
      REPETER
           ECRIRE(“Vous préférez: 1: Pikachu 2: Salamèche?”)
           LIRE(rep3)
      TANT QUE(rep3≠1 ET rep3≠2)
      …
      …

FIN
```

# On va essayer de faire mieux...

- J'ai un code qui s'applique exactement plusieurs fois dans mon programme, ouf, merci le copier/coller !
- J'ai un autre code, comme précédemment, que je peux répliquer, modulo la valeur d'une variable qui change… Il va falloir faire attention !
- Je dois modifier mon calcul, heureusement qu'il y a la recherche….

# On va essayer de faire mieux...

- Plutôt que d'écrire un algorithme en un seul bloc, nous allons assembler des morceaux de codes qui peuvent communiquer entre eux.

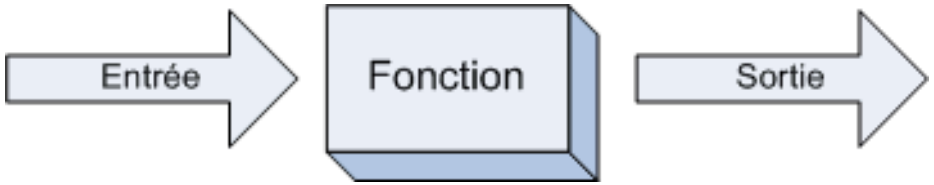
- On divise notre algorithme en utilisant des fonctions.

- Utiliser des fonctions permet non seulement de ne pas se répéter mais également d'avoir un programme plus clair.

# I. Principe des fonctions

# Fonction

- Une `FONCTION` est une suite d'instructions qui prend une liste de `paramètres` en entrée et produit (retourne) un `résultat` en sortie au code appelant cette fonction.



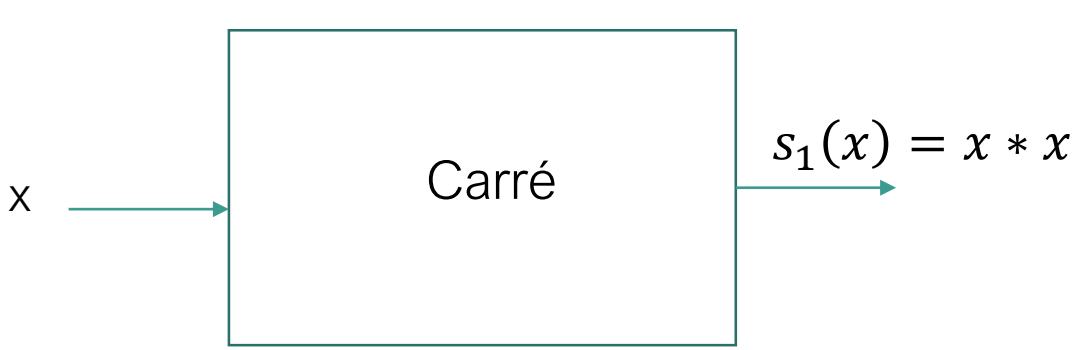

- Lorsqu'on appelle une fonction, il y a trois étapes.

1. L'entrée: on fait « rentrer » des informations dans la fonction (en lui donnant des informations avec lesquelles travailler).
2. Les calculs : grâce aux informations qu'elle a reçues en entrée, la fonction travaille.
3. La sortie : une fois qu'elle a fini ses calculs, la fonction renvoie un résultat.

   C'est ce qu'on appelle la sortie, ou encore le retour.

# Fonction

- Exemples de fonctions :





- Le concept de fonctions et de ses paramètres est similaire aux fonctions mathématiques : les paramètres sont des variables générales à remplacer lorsque l'on utilise la fonction.

$$s_1(x) = x * x \qquad s_2(a, b) = a^b$$

# Fonction

- Utilisation pratique des fonctions :



- Appeller la fonction : exécuter l'algorithme en remplacant les paramètres par des valeurs concrètes.
- Une fonction n'a qu'une seule sortie !
- Cette sortie est communiquée au reste du programme qui peut l'utiliser.

# II. Utilisation des fonctions

# Fonction

- Une **FONCTION** est une suite d'instructions qui prend une liste de **paramètres** en entrée et produit (retourne) UN UNIQUE **résultat** en sortie au code appelant cette fonction.
- Syntaxe :

```
FONCTION nomDeFonction(liste de paramètres) : type de retour
VARIABLES
    ...
    result : type de retour
DEBUT
    …
    RETOURNER result
FIN
```

# Un exemple complet

```
// Fonction qui élève un nombre au carré


FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

# Un exemple complet

```
// Fonction qui élève un nombre au carré
                          Liste paramètres
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

Zone de déclaration

Algorithme + retour

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN
```

Zone des fonctions

```
PROGRAMME testCarre
VARIABLE
        a, b : RÉEL
DÉBUT
        a ← 4
        b ← carre(a)
        écrire(a + "2 = " + res)
FIN
```

Programme principal

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, b : RÉEL
DÉBUT
        a ← 4
        b ← carre(a)
        écrire(a + "2 = " + res)
FIN
```

Un algorithme s'exécute **toujours** en commençant par le programme principal.

Le code des fonctions sera exécuté si les fonctions sont appelées.

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)
        écrire(a + "2 = " + res)
FIN
```



Eva ANSERMIN & Romuald GRIGNON

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)
        écrire(a + "2 = " + res)
FIN
```



Eva ANSERMIN & Romuald GRIGNON

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)
        écrire(a + "2 = " + res)
FIN
```

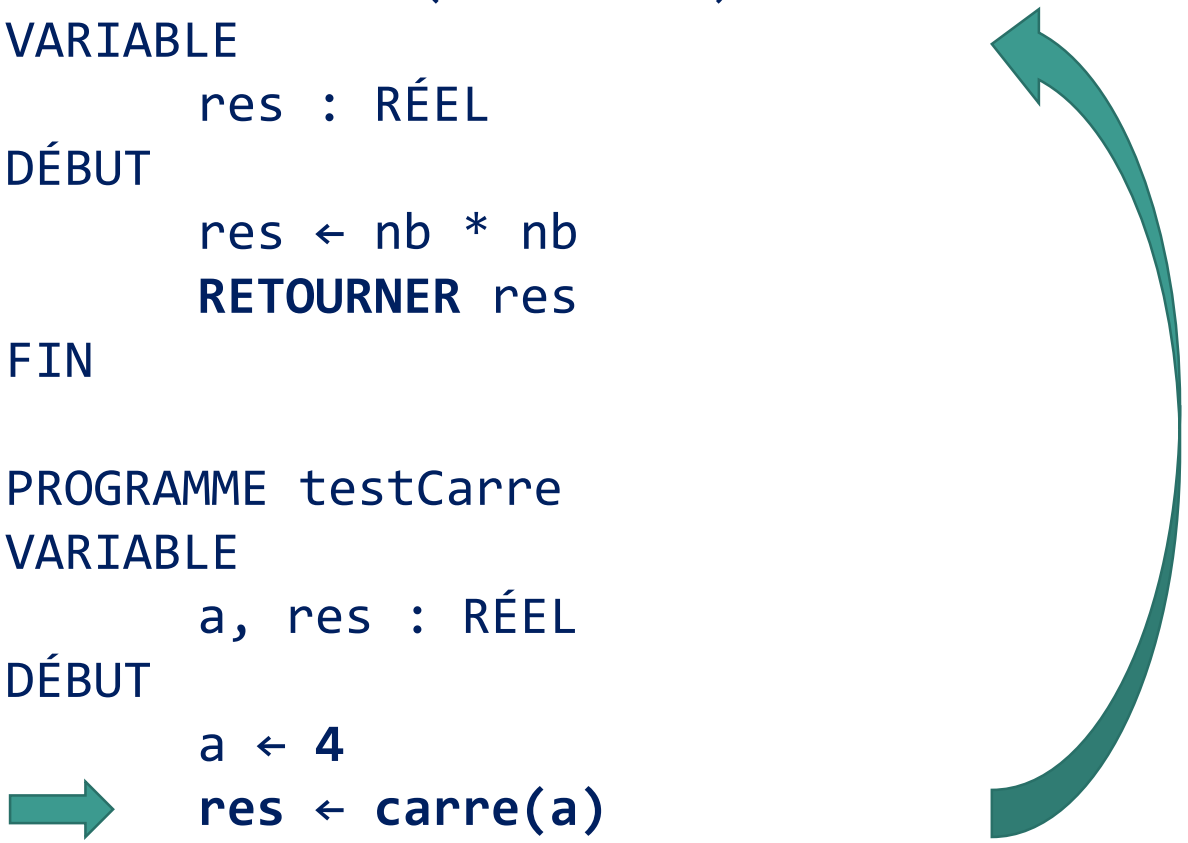
L' appel d'une fonction provoque l'arrêt du programme/fonction appelante et l'exécution du code de la fonction avec les paramètres.

Les paramètres de la fonction vont prendre les valeurs indiquées lors de cet appel.

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)
        écrire(a + "2 = " + res)
FIN
```



On appelle la fonction carré avec nb=4

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)
        écrire(a + "² = " + res)
FIN
```

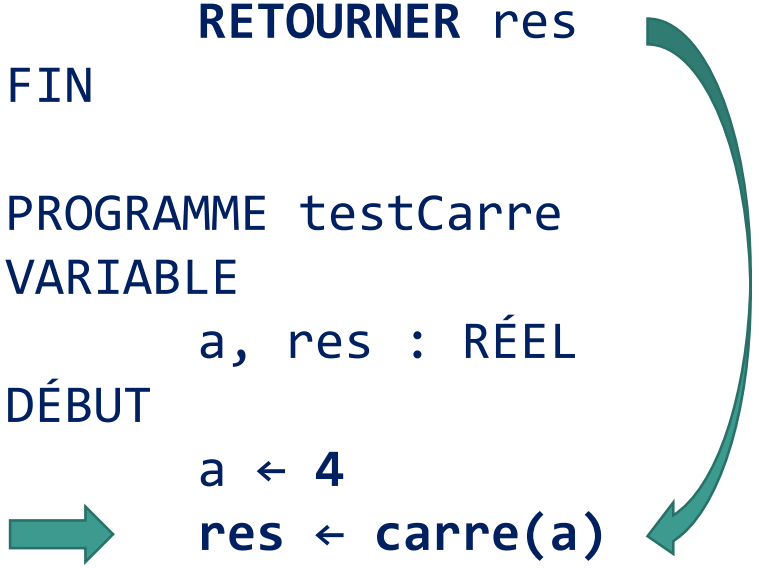


Eva ANSERMIN & Romuald GRIGNON

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)
        écrire(a + "2 = " + res)
FIN
```

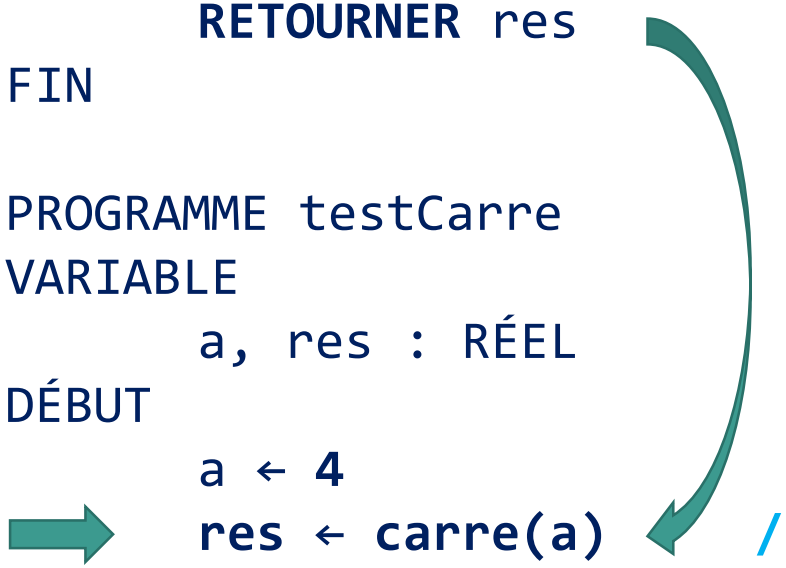


On retourne le resultat : carre(a) va etre remplacé par cette valeur retournée .

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)      // res<- 16
        écrire(a + "² = " + res)
FIN
```

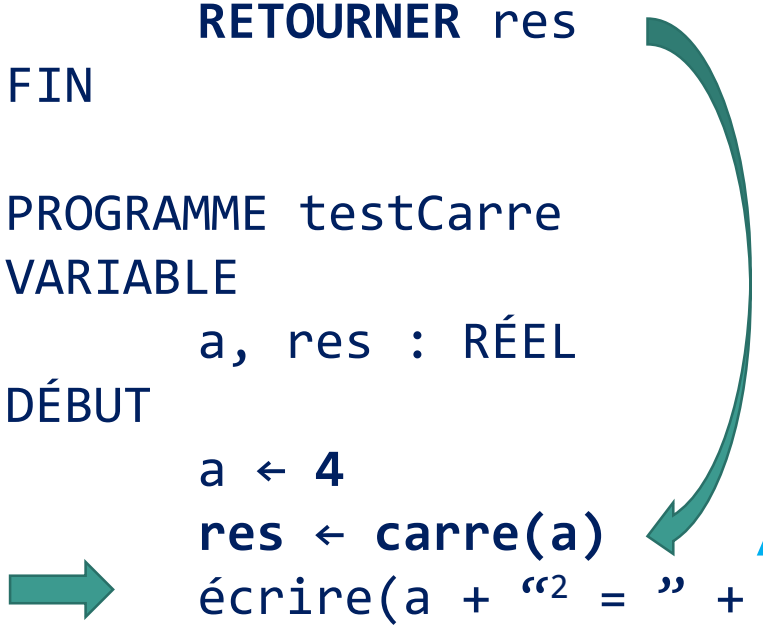


On reprend le programme normalement après l'appel de la fonction

# Appel et retour d'une fonction

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL
DÉBUT
        res ← nb * nb
        RETOURNER res
FIN

PROGRAMME testCarre
VARIABLE
        a, res : RÉEL
DÉBUT
        a ← 4
        res ← carre(a)      // res<- 16
        écrire(a + "² = " + res)
FIN
```



# Variables et fonctions

- Le programme principal peut être considéré comme la fonction qui s'exécute en premier dans l'algorithme.
- Les variables déclarées dans une fonction ne sont accessibles que dans celle-ci; on dit qu'elles sont **locales.** Lorsqu'on sort de la fonction, ces variables ne sont plus utilisables.
- Les seuls moyens de transmettre des données entre les différentes fonctions / programme principal sont donc :

  **Les paramètres** : pour donner des informations à la fonction appelée.

  L'instruction **RETOURNER** : pour transmettre le résultat de la fonction (une seule valeur).

# RETOURNER

- Le mot clef **RETOURNER** permet de transmettre une unique valeur au reste du programme : on ne retourne qu'**une** variable !
- L'instruction **RETOURNER** provoque un arrêt de la fonction et un retour à l'instruction appelante : toute les instructions qui suivent **RETOURNER** ne s'exécuteront jamais !

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
        ECRIRE ( « Instruction inutile ! »)
FIN
```

- Le type de retour doit être cohérent avec ce qui est retourné.

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN

FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

Zone des fonctions

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Programme principal

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables programme principal:

| nb | res |
|---|---|
| | |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables programme principal:

| nb | res |
|---|---|
| 3 | |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct cube:

| nb | res |
|---|---|
| 3 | |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct cube:

| nb | res |
|---|---|
| 3 | |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct carre:

| nb | res |
|---|---|
| 3 | |

Variables fct cube:

| nb | res |
|---|---|
| 3 | *'3 * carre(3)'* |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct carre:

| nb | res |
|---|---|
| 3 | |

Variables fct cube:

| nb | res |
|---|---|
| 3 | *'3 * carre(3)'* |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct carre:

| nb | res |
|---|---|
| 3 | 9 |

Variables fct cube:

| nb | res |
|---|---|
| 3 | *'3 * carre(3)'* |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct carre:

| nb | res |
|---|---|
| 3 | 9 |

Variables fct cube:

| nb | res |
|---|---|
| 3 | *'3 * carre(3)'* |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct cube:

| nb | res |
|---|---|
| 3 | *'3 * 9'* |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct cube:

| nb | res |
|---|---|
| 3 | 27 |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables fct cube:

| nb | res |
|---|---|
| 3 | 27 |

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'cube(3)'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN

FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN

PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables programme principal:

| nb | res |
|---|---|
| 3 | *'27'* |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN
```

```
FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN
```

```
PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

Variables programme principal:

| nb | res |
|---|---|
| 3 | 27 |

# Exemple d'execution:

```
FONCTION carre(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * nb
        RETOURNER res // on retourne le résultat
FIN

FONCTION cube(nb : RÉEL) : RÉEL
VARIABLE
        res : RÉEL // on déclare une variable résultat
DÉBUT
        res ← nb * carre(nb)
        RETOURNER res // on retourne le résultat
FIN


PROGRAMME testCube
VARIABLE
        nb, res : RÉEL
DÉBUT
        nb ← 3
        res ← cube(nb)
        écrire(res)
FIN
```

27

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables programme principal:

| x | y |
|---|---|
| | |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables programme principal:

| x | y |
|---|---|
| 10 | |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables fct soustraction:

| a | b | res |
|---|---|---|
| 15 | 10 | |

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables fct soustraction:

| a | b | res |
|---|---|---|
| 15 | 10 | |

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables fct soustraction:

| a | b | res |
|---|---|---|
| 15 | 10 | 5 |

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) ) // retour = 5
        ECRIRE( soustraction(x, y) )
FIN
```

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

5

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables fct soustraction:

| a | b | res |
|---|---|---|
| 10 | 15 | |

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables fct soustraction:

| a | b | res |
|---|---|---|
| 10 | 15 | |

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

Variables fct soustraction:

| a | b | res |
|---|---|---|
| 10 | 15 | -5 |

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) ) // retour = -5
FIN
```

Variables programme principal:

| x | y |
|---|---|
| 10 | 15 |

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

```
5
-5
```

# Fonction à plusieurs paramètres :

```
FONCTION soustraction(a,b : RÉEL) : RÉEL
VARIABLE
        res : réel
DÉBUT
        res ← a - b
        RETOURNER res
FIN
```

```
PROGRAMME TEST
VARIABLE
x,y : reel
DEBUT
        x ← 10
        y ← 15
        ECRIRE( soustraction(y, x) )
        ECRIRE( soustraction(x, y) )
FIN
```

C'est l'ordre des paramètres qui importe !!!

# III. Les procédures

# Procédure

- Une procédure est un ensemble d'instructions prenant en entrée des paramètres et **qui ne retourne rien.**
- Elles sont souvent utilisées **pour réaliser des affichages** ou réaliser des traitements sans effets de bord.
- Exemple simple :

```
PROCEDURE salutation(nom : chaîne)
DÉBUT
        ECRIRE("Bonjour"+nom+"!")
FIN
```

- Une procédure peut avoir des paramètres d'entrée : elle n'a juste pas d'instruction `RETOURNER` et s'arrête au mot clef `FIN`.

# Un exemple complet

```
PROCEDURE affMenu() // quand il n'a pas de paramètre, écrire ()
DÉBUT
    ECRIRE("     -- MENU --")
    ECRIRE("1 : Faire ceci")
    ECRIRE("2 : Faire cela")
    ECRIRE("3 : Configuration")
    ECRIRE("0 : Quitter")
    ECRIRE("Faites votre choix")
FIN

PROGRAMME Continuer
VARIABLES
     choix : entier
DÉBUT
      FAIRE     // pourquoi cette boucle ?
           affMenu()
           lire(choix)
      TANT QUE (choix < 0) ou (choix > 3)
      SELON choix :
           CAS 1: ...
           CAS 2: ...
           CAS 3: ...
           CAS 0: ...
      FIN SELON
FIN
```

```
-- MENU --
1 : Faire ceci
2 : Faire cela
3 : Configuration
0 : Quitter
Faites votre choix
|
```

# Et notre exemple de depart ?

```
PROGRAMME sondage
VARIABLES
      rep1, rep2, rep3 : entier
DÉBUT
      FAIRE
          ECRIRE(“Vous préférez: 1: les chats? 2: les chiens?”)
          LIRE(rep1)
      TANT QUE(rep1≠1 ET rep1≠2)
      FAIRE
          ECRIRE(“Vous préférez: 1: les pâtes? 2: le riz?”)
          LIRE(rep2)
      TANT QUE(rep2≠1 ET rep2≠2)
      FAIRE
          ECRIRE(“Vous préférez: 1: Pikachu? 2: Salamèche?”)
          LIRE(rep3)
      TANT QUE(rep3≠1 ET rep3≠2)
      …
      …

FIN
```

# Et notre exemple de depart ?

```
FONCTION ask(q : chaine):entier
VARIABLE
        rep :entier
        REPETER
                ECRIRE(q)
                LIRE(rep)
        TANT QUE(rep≠1 ET rep≠2)
        RETOURNER rep

PROGRAMME sondage
VARIABLES
     rep1, rep2, rep3 : entier
DÉBUT
     Rep1 <- ask(« Vous préférez: 1: les chats? 2: les chiens? »)
     Rep2 <- ask(« Vous préférez: 1: les pâtes? 2: le riz?     »)
     Rep3 <- ask(« Vous préférez: 1: Pikachu?   2: Salameche?  »)
     …
     …

FIN
```

# III. Langage C

# Procédures

```
void proc1(int a, float b) {
    instructions
}
```

```
Procédure proc1(a: entier,
b: réel) {
    instructions
}
```

# Procédures

- Utilisation du mot clé **void**
- Syntaxe :

```
void nomProc(paramètres) {

    instructions ;
}
```

```
void afficher(int n) {
    int i ;
    for (i=1; i<=n; i++) {
        printf("%d", i) ;
    }
}
```

# Fonctions

```
int ma_fonction(int a, float b) {
  …
  return (…) ;
}
```

```
Fonction ma_function(a: entier,
b:réel): entier
DEBUT
  …
  RETOURNER (…)
FIN
```

# Fonctions

- Syntaxe :

```
typeRetourné nomFonction(paramètres) {
    …
    return(…)
}
```

```
int factorielle(int n) {
    int f = 1 ;
    int i ;
    for (i=1; i<=n; i++) {
        f = f * i;
    }
    return f ;
}
```

# La fonction main

- En C le programme principal est vu comme une fonction/procédure. C'est une fonction qui s'appelle **`main`**. Elle peut même avoir des paramètres !
- C'est cette fonction qui sera parcourue lors de l'exécution du programme.

Mode fonction :

```
int main(){
    …
    return 0 ;
}
```

Mode procédure:

```
void main(){
    …
}
```

# Prototypage

- Le compilateur C lit les fichiers source de haut en bas. Il constatera donc une **erreur** si une fonction/procédure est déclarée **APRÈS** sa première utilisation. L'exemple suivant provoque une erreur de compilation.

```
#include <stdio.h>
int main () {
    printf("%d", toto(3)) ;
    return 0 ;
}

int toto (int n) {
    return 2 * n ;
}
```

# Prototypage

- Solution : déclarer les fonctions dans le bon ordre.

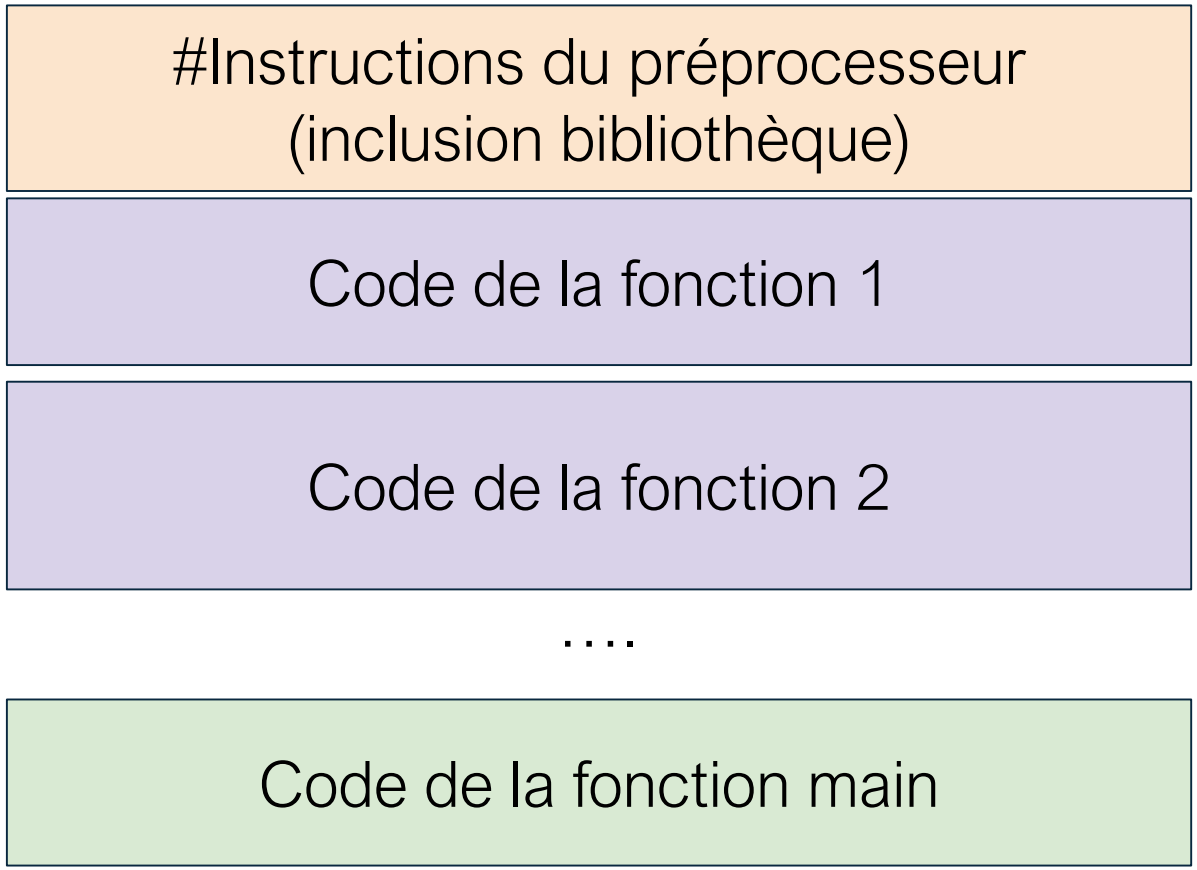
```
int toto (int n) {
    return 2 * n ;
}

int main () {
    printf("%d",toto(3)) ;
    return 0 ;
}
```

# Quelques fonctions utiles

- Les bibliothèques sont des recueils de fonctions déjà écrites que vous pouvez utiliser une fois la bonne bibliothèque incluse avec **`#define`**
- Fonctions mathématiques (dans **`math.h`**) :
  - `sqrt(float a)    : racine carrée de a`
  - `pow(int a, int b) : a puissance b`
  - `cos(float a), sin(float a), tan(float a)`
  - …
- Autres :
  - `rand() : génère un entier aléatoire (stdlib.h)`
  - `printf(…) affiche des caractères à l'écran (stdio.h)`
  - `scanf(…) permet de saisir des valeurs au clavier (stdio.h)`
  - …
- Le site koor.fr fourni une liste et des exemples d'utilisation des fonctions existantes en C.

# Structure générale d'un fichier .c



# Pour conclure...

- Objectifs de concision, clarté, indépendance et réutilisabilité.
- Une fonction = une fonctionnalité ; une fonction ne doit pas faire plus d'une vingtaine de lignes !
- Pensez à respecter les préconditions / à appeler correctement vos fonctions (nombres de paramètres, ordre, types, etc…).