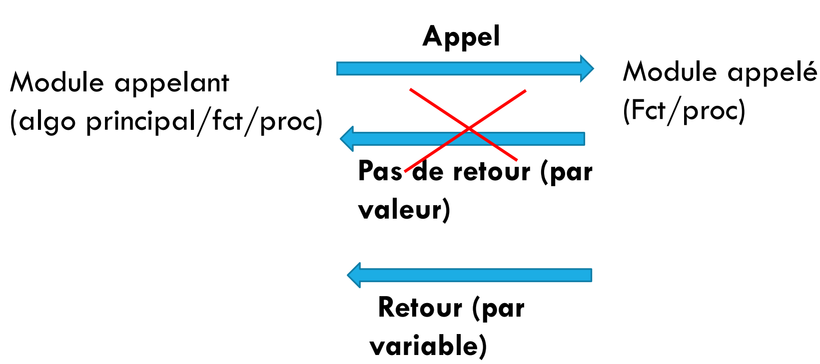
PARTIE 2.1
SOUS PROGRAMMES

ENSEIGNANTS:
MME. FEYROUZ HAMDAOUI, M. SOFIENE BEN AHMED ET M. MOEZ HAMMAMI

2025/2026 –SU1

SOUS PROGRAMMES: CONCEPTS DE BASE



Exemple:

X ← 5

Y ← 3

S ← Somme (x,y)

fonction Somme (a, b: entier): entier

Som ← a+b

retourner (Som)

SOUS PROGRAMMES: CONCEPTS DE BASE

Variable
- Variable global
- Variable locale

Paramètres
- Paramètres effectifs
- Paramètres fictifs

Module
- Fonction (1 seul résultat)
- Procédure (0 ou +eurs résultats)

Type de passage
- Passage par valeur
- Passage par variable

PROCÉDURE

Procédure *Nom_Procédure* ***[(paramètres)]***
-- Partie des déclarations
Début
 -- Corps de la procédure
Fin

(identificateur[, identificateur…] : type [; identificateur[, identificateur…] : type…])

5

FONCTION

Fonction *Nom_Fonction* ***(Arguments) :*** *Type_Résultat*
-- Partie des déclarations
Début
 …
 Nom_Fonction ←**…** *-- (a)*
 …
Fin

(identificateur[, identificateur…] **:** type [; identificateur[, identificateur…] **:**

6

ACTIVITÉ 9 (1/2)

Le programme présenté ci-dessous permet de décrypter un message en anglais. Vous êtes appelés à faire la trace d'exécution de ce programme pour décrypter le message stocké dans le tableau **T**.

Voici le contenu initial du tableau **T** (**taille = 5**)

"HXG\U'"	"\rx"	"lezi"	"7"	"swv"
0	1	2	3	4

```
DefType
    Tab = Tableau[1..5] de Chaîne
DefVar
    T: Tab = {"HXG\U'", "\rx", "lezi", "7", "swv"}

Procédure Decrypter(T: Tab, taille: Entier)
DefVar
    i: Entier
Début
    Pour i de 1 A taille Faire
        T[i] ← Decrypter_UnMot(T[i])
    FinPour
Fin
```


ACTIVITÉ 9 (2/2)

```
Fonction Decrypter_UnMot(mot: Chaîne): Chaîne
DefVar
    i(Entier)
    motDec (Chaîne)
    code, i (Entier)
Début
    motDec ← ""
    Pour i de 1 A Long(mot) Faire
        Code ← Asc(mot[i]) - Long(mot)
         motDec ← Concat(motDec, Chr(Code))
    FinPour
    Decrypter_UnMot  ← motDec
Fin
```

Quel sera le contenu du **T** après l'exécution de la procédure **Decrypter**

?	?	?	?	?
0	1	2	3	4

DÉFINITION EN C

- utile quand vous avez à faire le même type de calcul plusieurs fois dans le programme, mais avec des valeurs différentes.
- Typiquement, pour le calcul de l'intégrale d'une fonction mathématique f . Comme en mathématique, une fonction prend un ou plusieurs arguments entre parenthèses et renvoie une valeur.

9

DÉCLARATION (1/4)

```
type nom( type1 arg1 , type2 arg2, ... , typen argn) { */ prototype */
   déclaration variables locales;
   instructions;
   return (expression);
}
```

`type` est le type de la valeur renvoyée par la fonction. `type1` est le type du 1er argument `arg1` … . Les variables locales sont des variables qui ne sont connues qu'à l'intérieur de la fonction. `expression` est évaluée lors de l'instruction `return (expression);`, c'est la valeur que renvoie la fonction quand elle est appelée depuis `main()`.

10

DÉCLARATION (2/4)

La 1ère ligne de la déclaration est appelée le **prototype** de la fonction.
Exemple de fonction :

```
float affine( float x ) { /* la fonction 'affine' prend 1 argument réel */
                          /* et renvoie un argument réel */
    int a,b;
    a=3;
    b=5;
    return (a*x+b); /* valeur renvoyee par la fonction */
}
```


DÉCLARATION (3/4)

On n'est pas obligés de déclarer des variables locales :

```
float norme( int x, int y ){ /* la fonction 'norme' prend 2 arguments */
                             /* entiers et renvoie un résultat réel */
    return (sqrt(x*x+y*y)); /* sqrt est la fonction racine carree */
}
```

On peut mettre plusieurs instructions `return` dans la fonction :

```
float val_absolue( float x ) {
    if (x < 0) {
        return (-x);
    } else {
        return (x);
    }
}
```

DÉCLARATION (4/4)

Une fonction peut ne pas prendre d'argument, dans ce cas-là, on met à la place de la déclaration des arguments, le mot-clé `void` :

```
double pi( void ) { /* pas d'arguments */
   return(3.14159);
}
```

Une fonction peut aussi ne pas renvoyer de valeur, son type sera alors `void` :

```
void mess_err( void ) {
   printf("Vous n\'avez fait aucune erreur\n");
   return; /* pas d'expression après le return */
}
```


APPEL DE LA FONCTION (1/2)

- Une fonction f() peut être appelée depuis le programme principal main() ou bien depuis une autre fonction g() à la condition de rappeler le prototype de f() après la déclaration des variables de main() ou g() :

```
#include <stdio.h>
main(){
    int x,y,r;

    x=5;
    y=235;
    r=plus(x,y); /* appel d'une fonction avec arguments */
}
```

APPEL DE LA FONCTION (2/2)

```
int plus( int x, int y ){

   mess_err(); /* appel d'une fonction sans arguments */
   return (x+y));
}

void mess( void ) {
   printf("Vous n\'avez fait aucune erreur\n");
   return;
}
```

Quand le programme rencontre l'instruction return, l'appel de la fonction est terminé. Toute instruction située après lui sera ignorée.

PARTIE 2.2
RÉCURSIVITÉ

ENSEIGNANTS:
MME. FEYROUZ HAMDAOUI, M. SOFIENE BEN AHMED ET M. MOEZ HAMMAMI

2025/2026 –SU1

RÉCURSIVITÉ: CONCEPT ALGORITHMIQUE?

- Ascendant
- Suite mathématique
- Arbre au nature
- Algorithme récursif: Algorithme qui fait appel à lui-même

Exemple: fonction factorielle

RÉCURSIVITÉ: TYPE

Récursivité des objets

Récursivité des traitements

- Récursivité directe
- Récursivité indirecte

NOTIONS

Notion d'environnement	Terminaison	Valeur d'arrêt	Profondeur
• Pile d'environnement	• Tout appel récursif doit contenir une clause conditionnelle	• La valeur pour laquelle les appels récursifs s'arrêtent	• $F_i - f_t$

ITERATION OU RECURSIVITE

<u>*Fonction*</u> *Fibo (n : Entier) : Entier*

-- Précond : n≥ 0

<u>*Début*</u>

    <u>*Si*</u> *(n = 0 Ou n =1)* <u>*Alors*</u>

        Fibo ← n

      <u>*Sinon*</u>

        Fibo ← Fibo (n-2) + Fibo(n-1)

    <u>*Fin Si*</u>

<u>*Fin*</u>

ITERATION OU RECURSIVITE

- Acker (0,n) = n + 1
- Acker (m,0) = Acker (m-1,1) pour m ≥ 1
- Acker (m,n) = Acker(m-1, Acker (m,n-1)) pour m ≥ 1 et n ≥ 1

Fonction Acker (m,n : Entier) : Réel
-- Précond : m≥ 0 et n≥ 0
Début
  Si (m = 0) Alors
    Acker ← n + 1
  SinonSi (n = 0) Alors
    Acker ← Acker (m-1,1)
  Sinon
    Acker ← Acker (m-1, Acker (m, n-1))
  FinSi
Fin

DEFINITION D'ALGORITHMES RECURSIFS

- ❑Etape 1 : Paramétrage du problème
- ❑Etape 2 : Recherche d'un cas trivial ainsi que sa solution
- ❑Etape 3 : Décomposition du cas général

ACTIVITÉ 10

```
Fonction Deviner (a : Entier, n : Entier) : Entier
DefVar
  b(Entier)
Début
  Si (n = 1) Alors
     Deviner ← a
  Fin Si
  b ← Deviner(a, n Div 2)
  Si (n Mod 2 =0) Alors
    Deviner ← b + b
  Sinon
    Deviner ← b + b + a
  FinSi
Fin
```

23

```
Algorithme Principal
DefVar
  k1, k2, k3, k4 (Entier)
Début
  k1 ← Deviner(3,3)
  k2 ← Deviner(4,11)
  k3 ← Deviner(10,19062012)
  k4 ← Deviner(2,-5)
  Ecrire("k1 = ", k1, " k2 = ", k2, " k3 = ", k3, " & k4 = ", k4)
Fin
```

TRAVAIL DEMANDE :

1) Qu'est-ce qui sera affiché lors de l'exécution de l'algorithme **Principal** ?

2) Si on changeait la structure conditionnelle

`Si (n = 1) Alors Deviner ← a  FinSI`

par

`Si (n = 0) Alors Deviner ← 0  FinSI`

Dites quel serait l'effet sur les valeurs suivantes :

a) k1, k2, k3 et k4.

b) La valeur de **Deviner (a, n)** lorsque **n<0**.

24

RÉCURSIVITÉ MULTIPLE: TOUR DE HANOI

L'exemple le plus classique pour illustrer la méthode récursive est celui des ***Tours de Hanoi*** :

N disques sont empilés par ordre de diamètre décroissant (tous les disques sont de diamètres différents) sur un piquet A. Deux autres piquets B et C peuvent recevoir des disques, à condition que ceux-ci soient toujours empilés selon la même règle du diamètre décroissant. Le but du jeu est de transporter les N disques du piquet A au piquet C, en utilisant éventuellement le piquet B, tout en respectant les deux règles suivantes :

- Ne déplacer qu'un seul disque à la fois.
- Ne placer un disque que sur un disque de diamètre supérieur, ou un piquet libre.

25

RÉCURSIVITÉ MULTIPLE: TOUR DE HANOI

La démarche récursive :

- Paramétrage du problème :
  - N : nombre de disques
  - A, B, C : les piquets (départ, intermédiaire et arrivée)

- Recherche de la valeur d'arrêt :
  - Si (N = 1) Alors « Déplacer un disque de A vers C » : Déplacer (A, C)

- Décomposition du cas général : Résoudre le problème pour N disques ne présente pas de difficulté si on sait le résoudre pour (N-1) disques ; en effet, transporter les N disques du piquet A au piquet C, c'est :
  - Transporter les (N-1) premiers disques de A vers B (C piquet intermédiaire), puis
  - Déplacer le disque restant de A vers C, et enfin
  - Transporter les (N-1) disques de B vers C (A piquet intermédiaire).

RÉCURSIVITÉ MULTIPLE: TOUR DE HANOI

```
Procédure Hanoi (Don n : Entier ; Don A, B, C : Chaîne)
-- Précond : n ≥ 1
Début
   Si (n = 1)  Alors
        Ecrire ("Déplacer un disque de ", A, " vers ", C)
   Sinon
        Hanoi (n-1, A, C, B)
        Ecrire ("Déplacer un disque de ", A, " vers ", C)
        Hanoi (n-1, B, A, C)
   Fin Si
Fin
```


RÉCURSIVITÉ MULTIPLE: TOUR DE HANOI

Le nombre de déplacements est une fonction du nombre de disques, en effet :

- Si $n = 0 \Rightarrow$ nombre de déplacements = 0
- Si $n = 1 \Rightarrow$ nombre de déplacements = 1
- Si $n = 2 \Rightarrow$ nombre de déplacements = 3
- Si $n = 3 \Rightarrow$ nombre de déplacements = 7
- Si $n = 5 \Rightarrow$ nombre de déplacements = 31
- Si $n = 8 \Rightarrow$ nombre de déplacements = 255
- Si $n = 10 \Rightarrow$ nombre de déplacements = 1023
- ...
- $\forall$ **$n \geq 1$** $\Rightarrow$ nombre de déplacements **$= 2^n - 1$**