Algorithmique

Module 1 - Introduction à l'algorithmique

Objectifs

- Découvrir ce qu'est un algorithme
- Savoir distinguer le rôle du développeur de celui de la machine
- Comprendre les buts de l'algorithmique

Qu'est-ce que « l'Algo » ?

- La machine est capable de réaliser des opérations très vite…

  … mais elle est bête !

- L'intelligence est humaine !
  - C'est le programmeur qui « donne » de son intelligence à la machine

- À nous d'expliquer à la machine ce qu'elle doit faire pour résoudre un problème donné

Exemples « d'Algo » dans la vie de tous les jours

- Une recette de cuisine
  - Liste des ingrédients

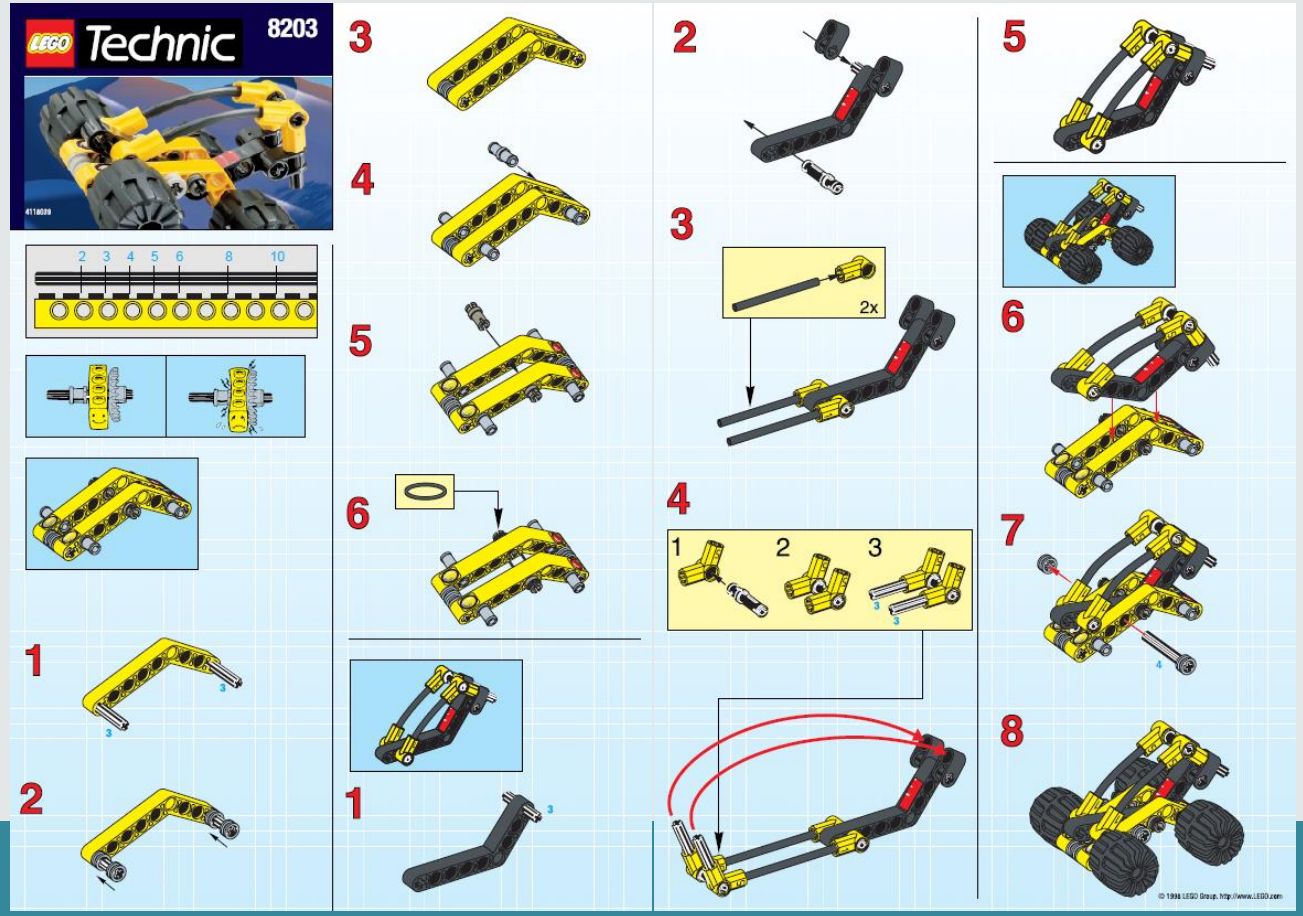
250 g de farine	1 pincée de sel	5 cl de rhum
4 œufs	50 grammes de beurre	
½ L de lait	1 sachet de sucre vanillé	

[www.pate-a-crepe.info]

  - Les étapes
    - Dans un saladier, verser la farine et les œufs.
    - Puis progressivement ajoutez le lait tout en mélangeant avec votre fouet.
    - Ajoutez le sucre vanillé, la pincée de sel.
    - Laisser reposer la pâte.
    - Versez une demi-louche de votre pâte à crêpe et faites cuire 1 à 2 minutes par face.
    - Conseil : servir avec du caramel au beurre salé (recette page suivante).

Exemples « d'Algo » dans la vie de tous les jours

- Une suite de dessins pour un jeu



[www.lego.com]

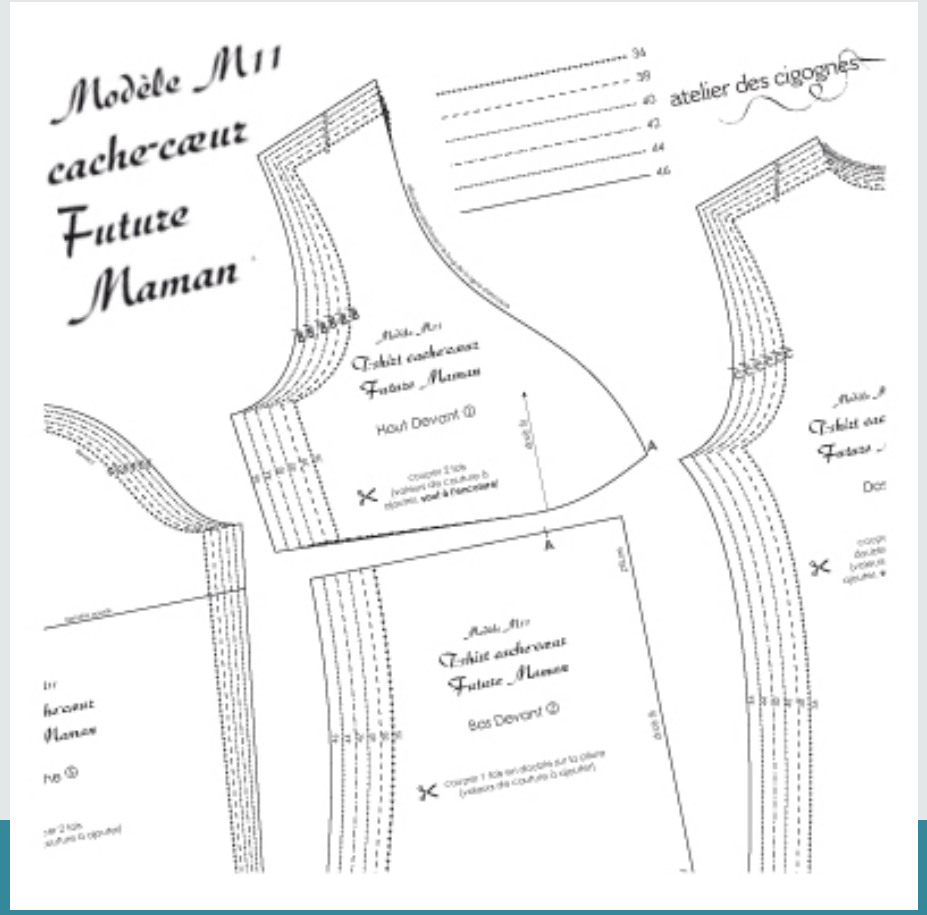
Exemples « d'Algo » dans la vie de tous les jours

- Une notice de montage

[michelin]

Exemples « d'Algo » dans la vie de tous les jours

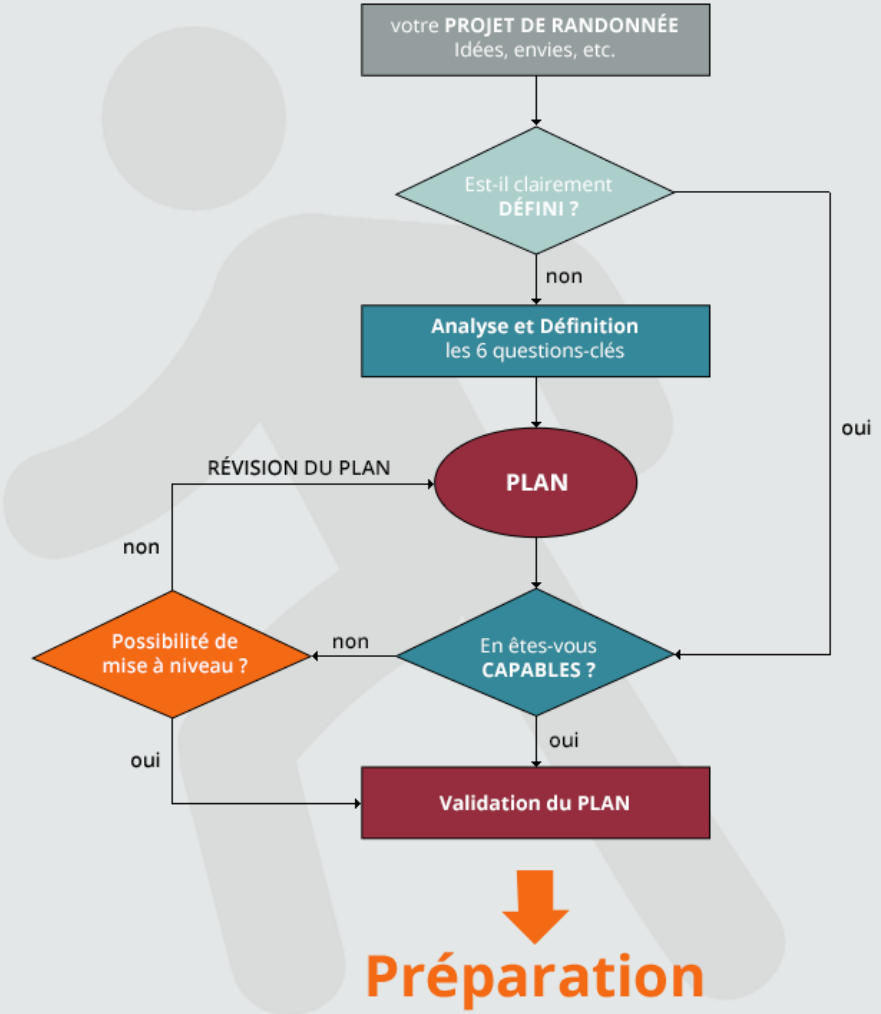
- Un patron



[atelierdescigognes.fr]

Exemples « d'Algo » dans la vie de tous les jours

- Un logigramme

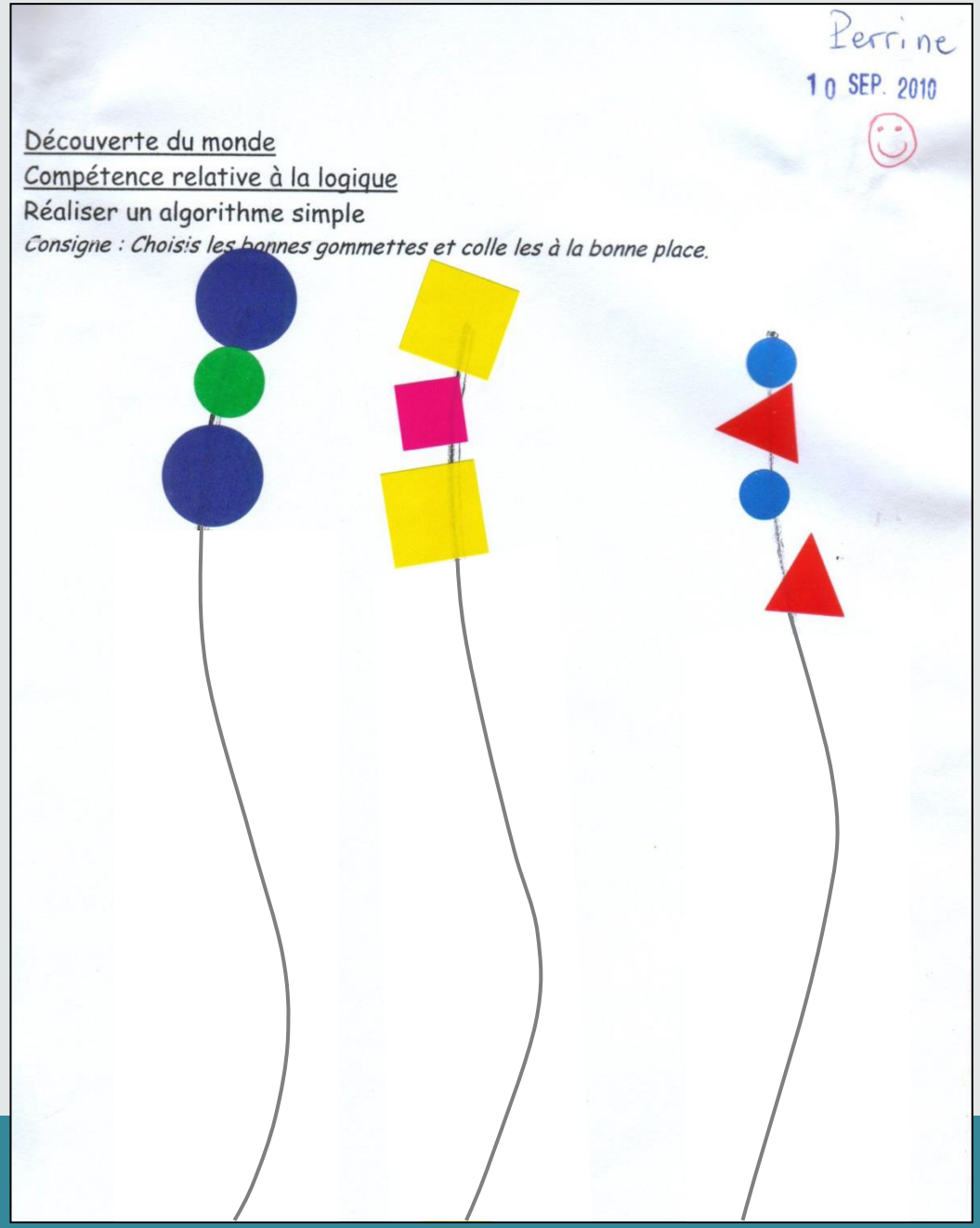


[d'après alaintrt.canalblog.com]

Introduction à l'algorithmique

Définition

- Une suite finie et non-ambiguë d'opérations permettant de donner la réponse à un problème



A notre tour !

- Écrire les étapes nécessaires pour prendre un café en salle de pause
  - Aller en salle de pause
  - Mettre des pièces dans la machine jusqu'à avoir un crédit suffisant
  - Appuyer sur la touche du café que vous souhaitez
  - Attendre que le café soit entièrement passé
  - Prendre son café
  - Boire son café
  - Revenir vite en cours avant la fin de la pause

Buts de l'algorithmique

- Conception : *quelle est la suite d'instructions permettant la résolution du problème posé ?*
- Complexité : *quel est l'algorithme le plus économe (en temps et en mémoire) ?*
- Calculabilité : *existe-t-il un algorithme permettant de résoudre le problème posé ?*
- Correction : *l'algorithme proposé répond-il au problème ?*

Pourquoi l'algorithmique ?

- L'algorithme est indépendant du langage de programmation et de la machine
- Pourquoi ne pas coder directement dans un langage de programmation ?
  - Pour réfléchir à la façon de procéder sans se soucier de contraintes techniques
  - Pour réfléchir entre programmeurs ne connaissant pas forcément les mêmes langages

Algorithmique

Module 2 - Les instructions de base en pseudo-code

Objectifs

- Découvrir l'une des manières possibles de décrire un algorithme
- Comprendre les opérations basiques réalisées par la machine
- S'approprier les instructions de base du pseudo-code
- Écrire ses premiers algorithmes en pseudo-code

Un exemple d'algorithme en pseudo-code

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Structure de l'algorithme

Nom de l'algorithme

Commentaire

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les déclarations

Les instructions

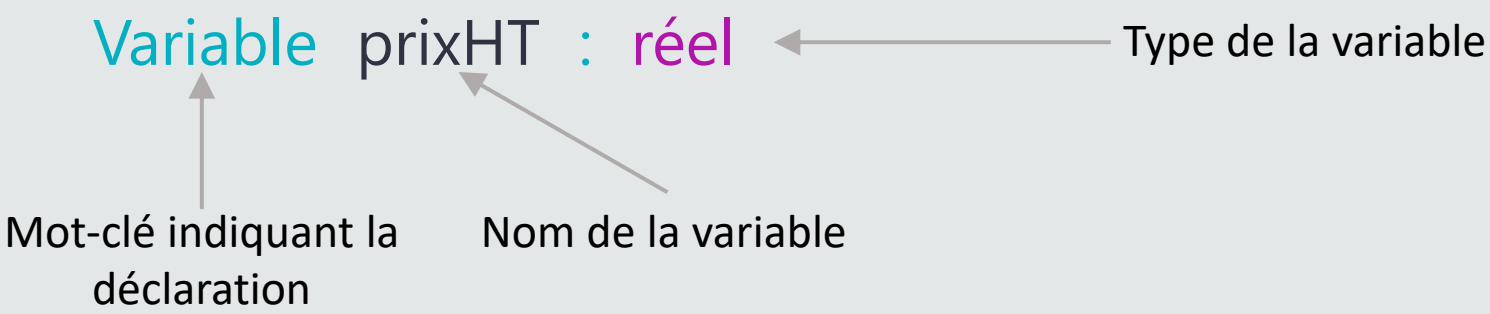
Les instructions de base en pseudo-code

Déclaration d'une variable

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Déclaration d'une variable



- Instruction permettant de réserver en mémoire de l'espace pour stocker une donnée
- Nom de la variable : nom sans espace commençant par une minuscule
- Types possibles : entier, réel, booléen, caractère, texte…

Les instructions de base en pseudo-code

Les types

Types	Exemples	Définition
entier	4; -2; 100 ...	Nombre entier positif ou négatif (ℤ)
réel	12,68; 3,14; 2 ...	Nombre à virgule positif ou négatif (ℝ)
booléen	VRAI; FAUX	Variable ne comportant que deux états : VRAI ou FAUX
caractère	'a'; 'ç'; '@'; ' '; '7' ...	1 seul caractère
texte	"bonjour"; "e"; ""; "test"; "78"...	Une suite de caractères

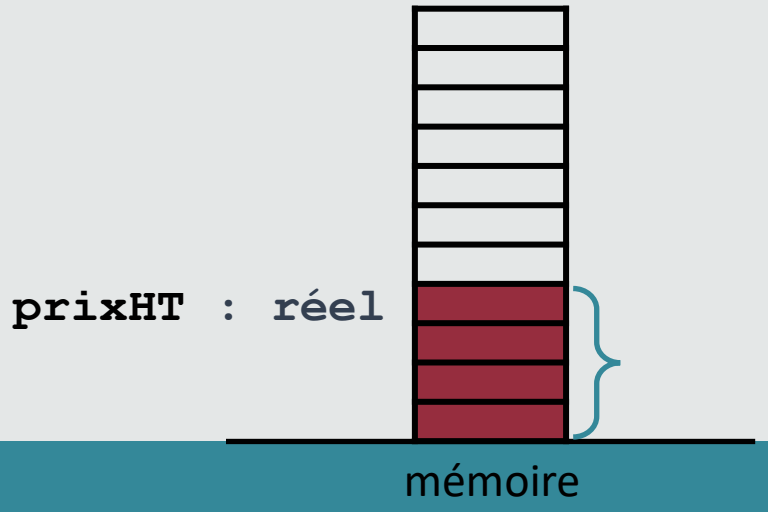
Les instructions de base en pseudo-code

Déclaration d'une variable

Variable prixHT : réel ← Type de la variable

Mot-clé indiquant la déclaration

Nom de la variable



Réservation d'espace dans la mémoire pour stocker la donnée

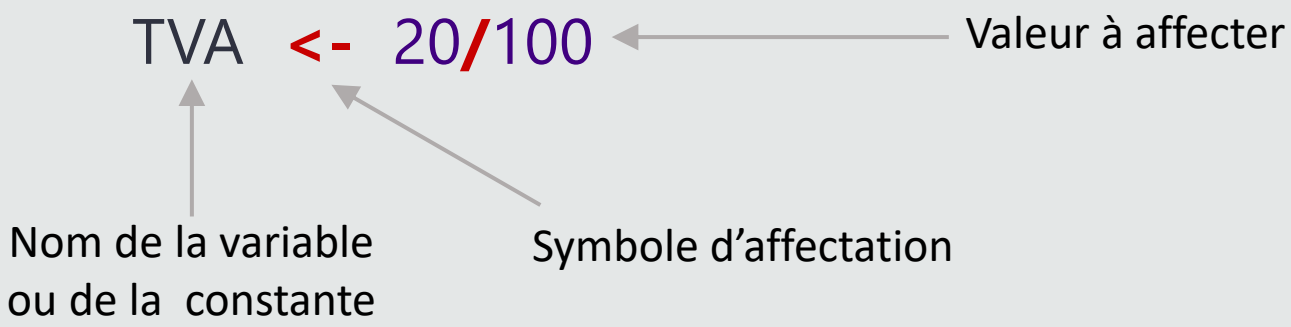
Les instructions de base en pseudo-code

Affectation d'une valeur

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Affectation d'une valeur



- Instruction permettant d'affecter une valeur à une variable ou à une constante
- Valeur à affecter : valeur ou calcul du même type que la variable ou la constante
- Pour une constante, l'affectation doit être faite en même temps que la déclaration

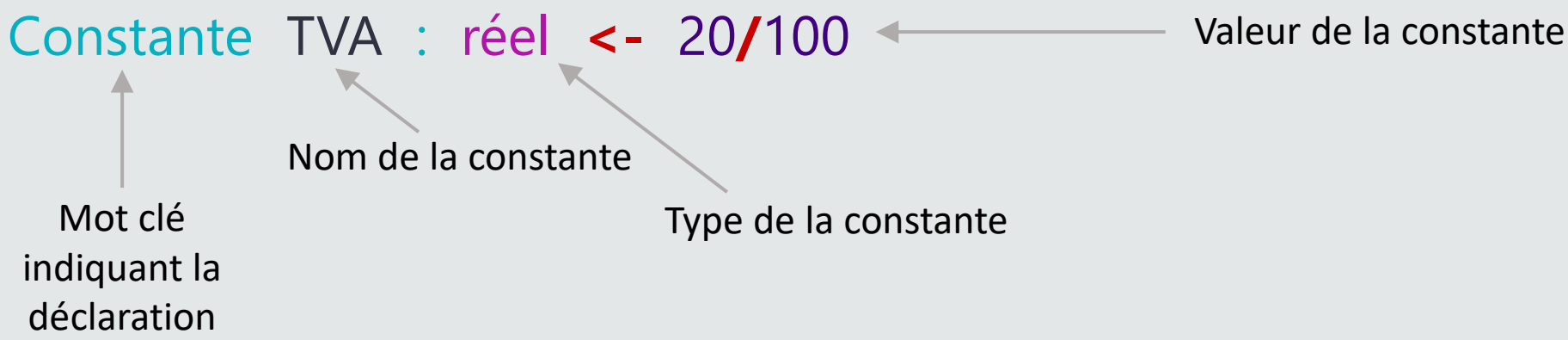
Les instructions de base en pseudo-code

Déclaration d'une constante

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Déclaration d'une constante



- Comme une variable sauf que la valeur ne change jamais
- Nom de la constante : nom sans espace écrit tout en majuscules
- Même type de données que pour une variable

Les instructions de base en pseudo-code

Calculs

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Calculs

20**/**100 prixHT*****(1**+**TVA**)**

- Opérateurs arithmétiques
  - **+** : addition
  - **-** : soustraction
  - ***** ou **×** : multiplication
  - **/** : division
  - **div** : division entière
  - **%** ou **mod** : reste de la division entière

Les instructions de base en pseudo-code

Calculs

1 < 20 / 100 17 ≠ 34 22 ≤ 22

- Opérateurs arithmétiques
- Opérateurs de comparaison
  - = : égal
  - ≠ : différent
  - < : inférieur
  - ≤ ou <= : inférieur ou égal
  - > : supérieur
  - ≥ ou >= : supérieur ou égal

Calculs

non(1<20/100) **et** (17≠34 **ou** 22≤22)

- Opérateurs arithmétiques
- Opérateurs de comparaison
- Opérateurs booléens

  et : la valeur de A et la valeur de B doivent être vraies pour que la valeur de « A **et** B » soit vraie

  ou : soit la valeur de A, soit la valeur de B, soit les deux valeurs doivent être vraies pour que la valeur de « A **ou** B » soit vraie

  non : si la valeur de A est vraie, la valeur de « **non** A » est fausse et réciproquement

Les instructions de base en pseudo-code

Affichage

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Affichage

```
écrire("Prix HT de l'article ?")
écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
```

Mot clé indiquant l'affichage

Suite d'éléments à afficher

- Permet d'afficher des informations à l'utilisateur
- Éléments possibles à afficher : constantes, variables, résultats de calcul, texte…

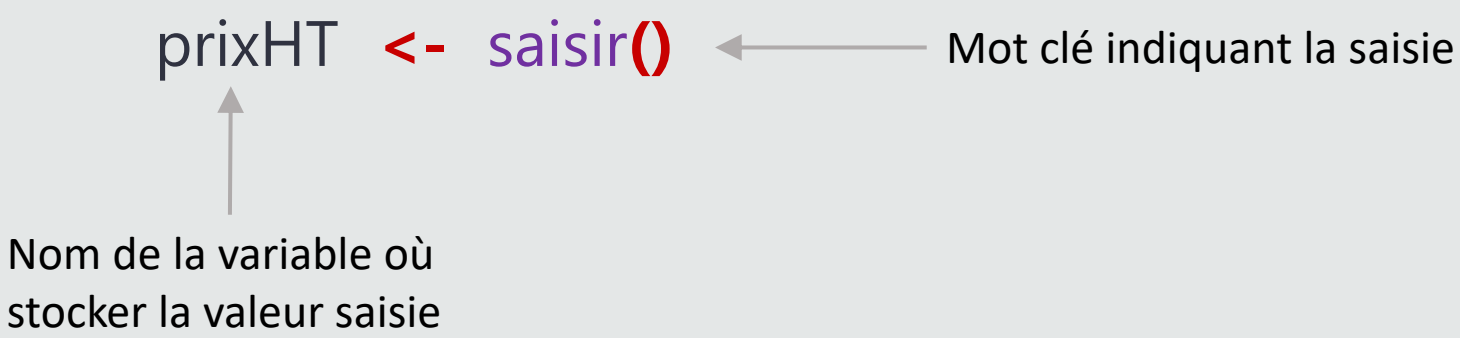
Les instructions de base en pseudo-code

Saisie

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

Les instructions de base en pseudo-code

Saisie



- Instruction permettant de stocker dans une variable la valeur saisie par l'utilisateur
- Saisie possible pour tous les types (entier, réel, booléen, caractère, texte…)
- Possibilité de combiner un affichage et une saisie

```
prixHT <- saisir("Prix HT de l'article ?")
```

équivalent à

```
écrire("Prix HT de l'article ?")
prixHT <- saisir()
```

Saisie

```
prixHT <- saisir()
21,25
```

Les instructions de base en pseudo-code

Un exemple d'algorithme en pseudo code

```
Algo tva
# calcule le prix TTC d'un article
Variable prixHT : réel
Constante TVA : réel <- 20/100
Début
    écrire("Prix HT de l'article ?")
    prixHT <- saisir()
    écrire("Prix TTC de l'article : " & prixHT*(1+TVA))
Fin
```

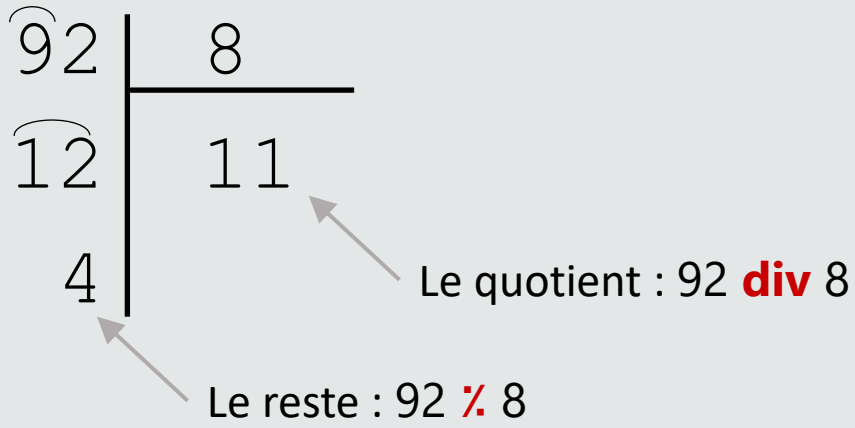
Les instructions de base en pseudo-code

TD : Quels affichages ?

```
Algo quelsAffichages
# Quels affichages produit cet algorithme ?
Variable valeur1, valeur2 : entier
Variable chaine1 : texte
Constante CST : réel <- 49,78
Début
    valeur1 <- 92 % 8
    valeur2 <- 2 * valeur1
    chaine1 <- "Test"
    écrire(chaine1 & " , valeur2 = " & valeur2)
    écrire(valeur1 & " # " & CST)
Fin
```

TD : Quels affichages ?

- Rappel sur la division entière



Les instructions de base en pseudo-code

TD : Quels affichages ?

Test , valeur2 = 8
4 # 49,78

Simulation de l'algorithme

Algorithme	valeur1	valeur2	chaine1
Algo quelsAffichages # Quels affichages produit cet algorithme ? Variable valeur1**,** valeur2 : entier Variable chaine1 : texte Constante CST : réel **<-** 49**,**78 **Début**			
valeur1 **<-** 92 **%** 8	**4**		
valeur2 **<-** 2 ***** valeur1	**4**	**8**	
chaine1 **<-** "Test"	**4**	**8**	**"Test"**
écrire**(**chaine1 **&** " , valeur2 = " **&** valeur2**)** écrire**(**valeur1 **&** " # " **&** CST**)** **Fin**	**4**	**8**	**"Test"**

Les instructions de base en pseudo-code

TD : Il fait quoi ?

```
Algo ilFaitQuoi
# Que réalise cet algorithme ?
Variable valeur1, valeur2 : réel
Début
    valeur1 <- saisir("Entrez valeur : ")
    valeur2 <- saisir("Entrez autre valeur : ")
    # traitement
    valeur1 <- valeur2
    valeur2 <- valeur1
    écrire("valeur1 = " & valeur1 & "; valeur2 = " & valeur2)
Fin
```

Les instructions de base en pseudo-code

TD : Il fait quoi ?

valeur1 = 22,9; valeur2 = 22,9

Simulation de l'algorithme

Algorithme	valeur1	valeur2
Algo ilFaitQuoi # Que réalise cet algorithme ? Variable valeur1**,** valeur2 : réel **Début** valeur1 **<-** saisir**(**"Entrez valeur : "**)**	**17,4**	
valeur2 **<-** saisir**(**"Entrez autre valeur : "**)**	**17,4**	**22,9**
# traitement valeur1 **<-** valeur2	**22,9**	**22,9**
valeur2 **<-** valeur1	**22,9**	**22,9**
écrire**(**"valeur1 = " **&** valeur1 **&** "; valeur2 = " **&** valeur2**)** **Fin**	**22,9**	**22,9**

Les instructions de base en pseudo-code

Comment faire pour échanger les valeurs de deux variables?

```
Algo inversion
# Saisit 2 valeurs et permute les variables
Variable valeur1, valeur2, tmp : réel
Début
    valeur1 <- saisir("Entrez valeur : ")
    valeur2 <- saisir("Entrez autre valeur : ")
    tmp <- valeur1
    valeur1 <- valeur2
    valeur2 <- tmp
    écrire("valeur1 = " & valeur1 & "; valeur2 = " & valeur2)
Fin
```

Les instructions de base en pseudo-code

Utilisation de notepad++

Démonstration

Les instructions de base en pseudo-code

TD : Vitesse moyenne

- Écrire un algorithme qui calcule la vitesse moyenne
  - Affichages et *saisies*

Saisissez la distance parcourue (km)
370
Saisissez le temps de parcours (min)
240
Vous vous êtes déplacé à une vitesse de 92,5 km/h

Les instructions de base en pseudo-code

TD : Vitesse moyenne

```
Algo vitesseMoyenne
# Calcule la vitesse moyenne sur un trajet.
Variable vitesse, min, km : réel
Constante UNE_HEURE : entier <- 60
Début
    km <- saisir("Saisissez la distance parcourue (km) : ")
    min <- saisir("Saisissez le temps de parcours (min) : ")
    vitesse <- km × UNE_HEURE / min
    écrire("vous vous êtes déplacé à " & vitesse & " km/h.")
Fin
```

Algorithmique

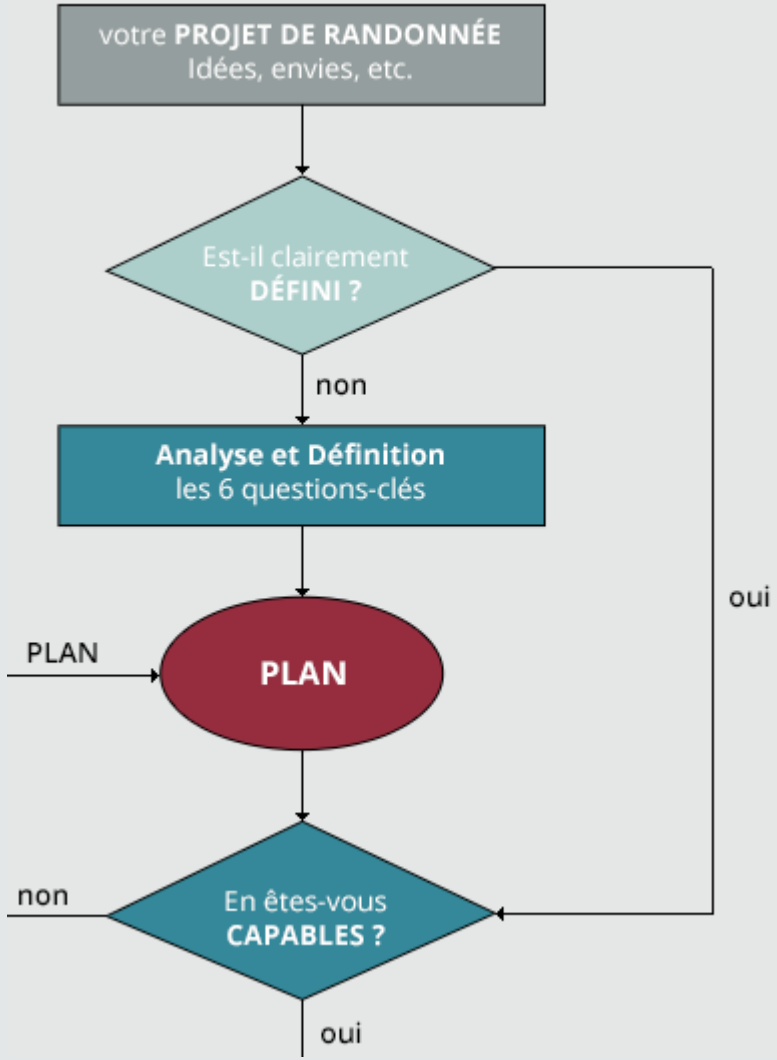
Module 3 - Les instructions conditionnelles

Objectifs

- Comprendre le principe des instructions conditionnelles
- Savoir utiliser les structures conditionnelles
- Appréhender des algorithmes de plus grande taille

Les instructions conditionnelles

Introduction



Les instructions conditionnelles

Le test Si : forme simple

```
Si condition_booléenne Alors
        suite_d_instructions
FSi
```

- Si la condition_booléenne vaut VRAI alors la suite_d_instructions est exécutée
  - Exemple

```
age <- saisir("Quel est votre âge ?")
Si age ≥ 18 Alors
        écrire("Vous êtes majeur !")
FSi
```

TD : Algorithme de météo

- Écrire un algorithme qui affiche « risque de verglas » si la température saisie est inférieure à 2° C

Les instructions conditionnelles

TD : Algorithme de météo

```
Algo risqueVerglas
# Demande à l'utilisateur de saisir la température et affiche une info
Variable temperature : entier
Constante LIMITE_VERGLAS : entier <- 2
Début
    temperature <- saisir("Entrez la température : ")
    Si temperature < LIMITE_VERGLAS Alors
        écrire("Risque de verglas")
    FSi
Fin
```

Le test Si : forme double

Si condition_booléenne **Alors**

suite_d_instructions

Sinon

autres_instructions

FSi

- Si la condition_booléenne vaut VRAI alors la suite_d_instructions est exécutée, dans le cas contraire ce sont les autres_instructions qui sont exécutées.

Le test Si : forme double

- Exemple

```
age <- saisir("Quel est votre âge ? ")
Si age ≥ 18 Alors
        écrire("Vous êtes majeur !")
Sinon
        écrire("Vous êtes mineur !")
FSi
```

TD : Algorithme de météo (version 2)

Température t	Message
$t < 2$	Risque de verglas
$2 \leq t < 15$	C'est pas chaud
$15 \leq t < 30$	Bonne température
$t \geq 30$	Trop chaud !

Les instructions conditionnelles

TD : Algorithme de météo (version 2)

```
Algo risqueVerglasV2
# Saisit la température et affiche une info
Variable temperature : entier
Constante LIMITE_VERGLAS : entier <- 2
Constante LIMITE_FROID : entier <- 15
Constante LIMITE_BIEN : entier <- 30
Début
    temperature <- saisir("Entrez la température : ")
    Si temperature < LIMITE_VERGLAS Alors
        écrire("Risque de verglas")
    Sinon
        Si temperature < LIMITE_FROID Alors
            écrire("Ce n'est pas chaud")
        Sinon
            Si temperature < LIMITE_BIEN Alors
                écrire("Bonne température")
            Sinon
                écrire("Trop chaud !")
            FSi
        FSi
    FSi
Fin
```

Le test Selon

```
Selon variable
        cas valeurs : instructions
        cas autres_valeurs : instructions
        cas encore_autres_valeurs : instructions
        ...
        autre : instructions
FSelon
```

L'instruction **Selon** permet de faciliter l'écriture s'il y a plus de deux choix

Les instructions conditionnelles

Le test Selon

- Exemple

```
Selon route
        cas "en ville" : vmax <- 50
        cas "hors agglo" : vmax <- 80
        cas "autoroute" : vmax <- 130
FSelon
```

Equivalent avec des **Si** :

```
Si route = "en ville" Alors
        vmax <- 50
Sinon
        Si route = "hors agglo" Alors
                vmax <- 80
        Sinon
                Si route = "autoroute" Alors
                        vmax <- 130
                FSi
        FSi
FSi
```

TD : Le nom du mois

- Écrire le nom du mois en toutes lettres en fonction de son numéro (saisi par l'utilisateur)

Les instructions conditionnelles

TD : Le nom du mois

```
Algo nomDuMois
# Demande la saisie d'un numéro de mois et affiche le nom du mois
Variable numMois : entier
Début
    numMois <- saisir("Entrez le numéro du mois : ")
    Selon numMois
        cas 1 : écrire("Janvier")
        cas 2 : écrire("Février")
        cas 3 : écrire("Mars")
        cas 4 : écrire("Avril")
...
        cas 10 : écrire("Octobre")
        cas 11 : écrire("Novembre")
        cas 12 : écrire("Décembre")
        autre : écrire("Saisie Incorrecte")
    FSelon
Fin
```

Les instructions conditionnelles

TD : Temps de cuisson

- Afficher le temps de cuisson d'une viande en fonction du type de la viande, du mode de cuisson et du poids de la viande
  - Pour cuire 500 grammes de bœuf, il faut
    - 10 minutes si on le veut BLEU
    - 17 minutes si on le veut A POINT
    - 25 minutes si on le veut BIEN CUIT
  - Pour cuire 400 grammes d'agneau, il faut
    - 15 minutes si on le veut BLEU
    - 25 minutes si on le veut A POINT
    - 40 minutes si on le veut BIEN CUIT
  - Le temps de cuisson est proportionnel au poids
  - Le résultat est affiché en secondes

Les instructions conditionnelles

TD : Temps de cuisson

```
Algo Cuisson
# Indique le temps de cuisson
Variable viande : entier
Variable poids : entier
Constante BOEUF : entier <- 1
Constante BLEU : entier <- 1
Constante BIEN_CUIT : entier <- 3
Constante BLEU_B : réel <- 10/500
Constante A_PT_B : réel <- 17/500
Constante B_CU_B : réel <- 25/500
Variable cuisson : entier
Variable coefficient : réel
Constante AGNEAU: entier <- 2
Constante A_POINT : entier <- 2
Constante UNE_MINUTE: entier <- 60
Constante BLEU_A : réel <- 15/400
Constante A_PT_A : réel <- 25/400
Constante B_CU_A : réel <- 40/400
Début
    écrire("Viande ?")                          # Choix du type de viande
    écrire(BOEUF & " – Bœuf")
    écrire(AGNEAU & " – Agneau")
    viande <- saisir()
    écrire("Cuisson ?")                         # Choix du type de cuisson
    écrire(BLEU & " – Bleu")
    écrire(A_POINT & " – A point")
    écrire(BIEN_CUIT & " – Bien cuit")
    cuisson <- saisir()
    poids <- saisir("Poids en gramme ?")        # Choix du poids de la viande
```

Viande ?
1 – Bœuf
2 – Agneau

TD : Temps de cuisson

```
Si viande = BOEUF Alors
    Selon cuisson
        cas BLEU :      coefficient <- BLEU_B
        cas A_POINT :   coefficient <- A_PT_B
        autre :         coefficient <- B_CU_B
    FSelon
Sinon
    Selon cuisson
        cas BLEU :      coefficient <- BLEU_A
        cas A_POINT :   coefficient <- A_PT_A
        autre :         coefficient <- B_CU_A
    FSelon
FSi
écrire("Le temps de cuisson est de " & poids × coefficient × UNE_MINUTE & " secondes")
Fin
```

Les instructions conditionnelles

TD : Bulletin de paie

- Afficher le bulletin de paie simplifié d'un salarié à partir des informations saisies
  - Le salaire de base
    - Les 169 premières heures sont payées sans majoration
    - Entre 169 et 180 heures une majoration de 50 % est appliquée
    - Au-delà de 180 heures la majoration passe à 60 %
  - La prime familiale
    - 1 enfant : 20 €
    - 2 enfants : 50 €
    - Au-delà de 2 enfants : 70 € +20 € par enfant supplémentaire

Les instructions conditionnelles

TD : Bulletin de paie simplifié

- Afficher le bulletin de paie d'un salarié à partir des informations saisies
  - Affichages et *saisies*

Nom de la personne ?

Duchemin

Prénom de la personne ?

Gérard

Statut ?

1 – Agent de service

2 – Employé de bureau

3 – Cadre

1

Nombre d'heures travaillées

190

Taux horaire ?

9,76

Nombre d'enfants ?

4

Bulletin de Gérard Duchemin

Statut : Agent de service

Salaire brut : 1966,64 € (169 h sans majoration, 11 h avec une majoration de 50 %, 10 h avec une majoration de 60 %)

Calcul des cotisations :

- Contribution pour le remboursement de la dette sociale et contribution sociale généralisée imposable

1966,64 € × 3,49 % = 68,64 €

- Contribution sociale généralisée non imposable

1966,64 € × 6,15 % = 120,95 €

- Assurance maladie

1966,64 € × 0,95 % = 18,68 €

- Assurance vieillesse

1966,64 € × 8,44 % = 165,98 €

- Assurance chômage

1966,64 € × 3,05 % = 59,98 €

- Retraite complémentaire (IRCEM)

1966,64 € × 3,81 % = 74,93 €

- Cotisation AGFF

1966,64 € × 1,02 % = 20,06 €

Total des cotisations salariales : 529,22 €

Salaire net : 1437,42 €

- Prime familiale : 110 €

Salaire net à payer : 1547,42 €

Algorithmique

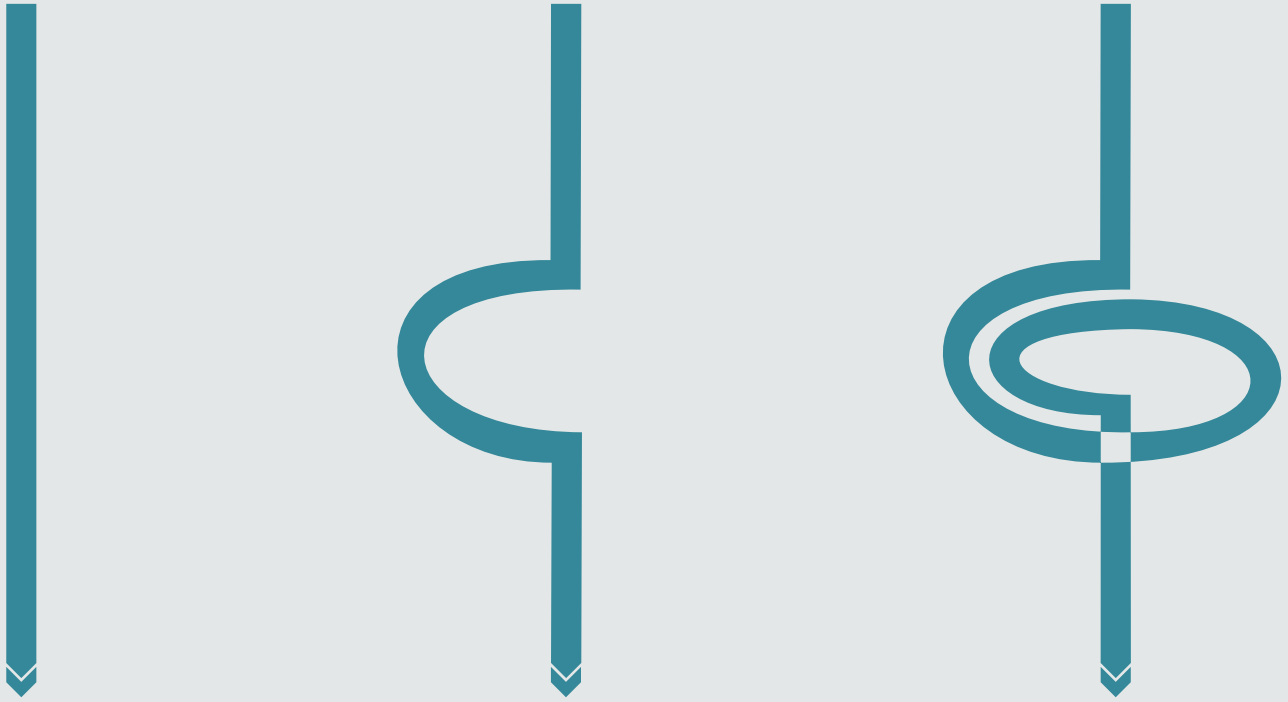
Module 4 - Les instructions itératives

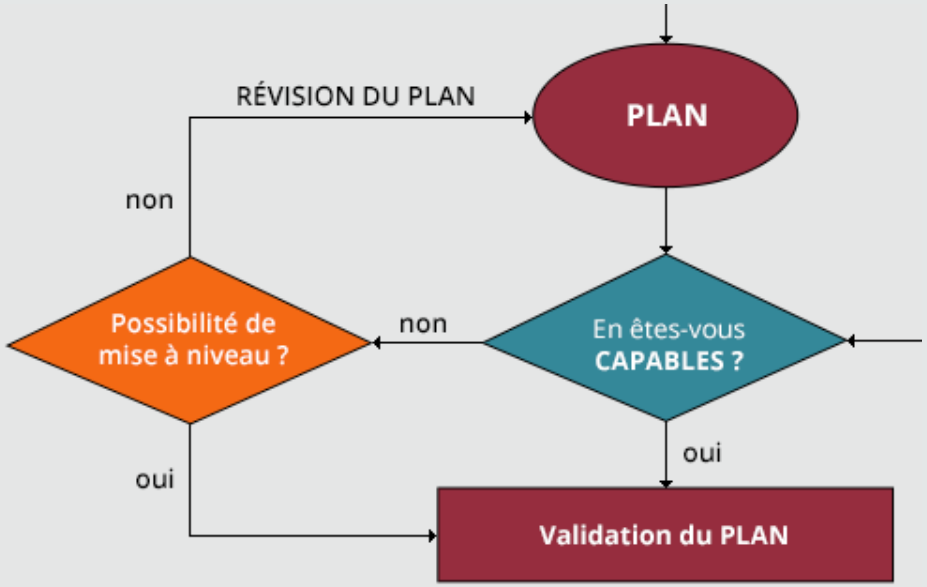
Objectifs

- Comprendre le principe des instructions itératives
- Savoir utiliser les structures itératives
- Connaître les caractéristiques de chaque boucle et savoir laquelle utiliser en fonction de la situation
- Appréhender les boucles imbriquées

Les instructions itératives

Introduction





Les instructions itératives

La boucle Pour

```
Pour var <- ini à fin [par pas]
        instructions
FPour
```

- Répète les instructions un certain nombre de fois
  - Exemple :

```
Pour i <- 1 à 3
        écrire("Passage n°" & i)
FPour
```

```
Passage n°1
Passage n°2
Passage n°3
```

TD : Moyenne de notes

- Affichages et *saisies*

Entrez le nombre de valeurs :
3
Valeur :
12
Valeur :
17,5
Valeur :
14
La moyenne est 14,5

TD : Moyenne de notes

```
Algo moyenneDesNotes
# Demande des valeurs et calcule de la moyenne
Variable nbVal, cpt : entier
Variable moyenne, saisie : réel
Début
    moyenne <- 0
    nbVal <- saisir("Entrer le nombre de notes : ")
    Si nbVal ≤ 0 Alors
        écrire("Le nombre de valeurs doit être strictement positif")
    Sinon
        Pour cpt <- 1 à nbVal
            saisie <- saisir("Valeur : ")
            moyenne <- moyenne + saisie / nbVal
        FPour
        écrire("La moyenne est " & moyenne)
    FSi
Fin
```

Les instructions itératives

La boucle TantQue

amorçage

TantQue condition

instructions

relance

FTq

- Répète les instructions tant que la condition vaut VRAI
- Les instructions d'amorçage permettent d'initialiser la variable sur laquelle porte la condition
- La relance permet de mettre à jour la variable sur laquelle porte la condition

Exemple

```
#Amorçage
saisie <- saisir("Quelle est la capitale de la France ?")
          #Condition
TantQue saisie ≠ "Paris"
      #Traitement
      écrire("Vous vous êtes trompé ! Réessayez…")
      #Relance
      saisie <- saisir("Capitale de la France ? ")
FTq
```

```
Quelle est la capitale de la France ? Lyon
Vous vous êtes trompé ! Réessayez…
Capitale de la France ? Nantes
Vous vous êtes trompé ! Réessayez…
Capitale de la France ? Paris
```

TD : Moyenne de notes (version 2)

- Saisir des notes jusqu'à ce que l'utilisateur saisisse -1 et en calculer la moyenne

Note (-1 pour terminer) ?
9
Note (-1 pour terminer) ?
18
Note (-1 pour terminer) ?
-1
La moyenne des notes est de 13,5

TD : Moyenne de notes (version 2)

```
Algo MoyenneDesNotesV2
#Calcule la moyenne des notes saisies par l'utilisateur
#L'utilisateur saisit -1 pour finir la saisie
Variable nbVal : entier <- 0
Variable saisie, total : réel
Constante STOP : entier <- -1
Début
    total <- 0
    saisie <- saisir("Note (" & STOP & "pour terminer) ?")
    TantQue saisie ≠ STOP
        total <- total + saisie
        nbVal <- nbVal + 1
        saisie <- saisir("Note (" & STOP & "pour terminer) ?")
    FTq
    Si nbVal ≠ 0 Alors
        écrire("La moyenne des notes est de " & total/nbVal)
    Sinon
        écrire("Aucune note n'a été saisie")
    FSi
Fin
```

TD : Moyenne de notes (version 3)

- Modifier votre algorithme pour afficher en plus le pourcentage de notes au-dessus de 10
  - Exemple

Il y a 50 % des notes au-dessus de 10/20.

TD : Moyenne de notes (version 3)

```
Algo MoyenneDesNotesV3      #Calcule la moyenne et donne le pourcentage de note ≥ 10
Variable nbVal <- 0, nbS <- 0 : entier          Variable saisie, total <- 0 : réel
Constante STOP : entier <- -1                   Constante PALIER : entier <- 10
Début
    saisie <- saisir("Note (" & STOP & "pour terminer) ?")
    TantQue saisie ≠ STOP
        total <- total + saisie
        nbVal <- nbVal + 1
        Si saisie ≥ PALIER Alors
            nbS <- nbS + 1
        FSi
        saisie <- saisir("Note ? ")
    FTq
    Si nbVal ≠ 0 Alors
        écrire("Moyenne = " & total/nbVal & " (" & nbS/nbVal×100 & "% ≥" & PALIER & ")")
    Sinon
        écrire("Aucune note n'a été saisie")
    FSi
Fin
```

TD : Devinez à quel nombre je pense

- Affichages et *saisies*

```
À quel nombre entre 1 et 100 je pense ?
23
C’est moins !
15
C’est plus !
19
C’est moins !
17
Bravo ! Vous avez trouvé !
```

TD : Devinez à quel nombre je pense

```
Algo nombreCacheCache
Constante NB_MYSTERE : entier <- aléa(1,100) #initialisation aléatoire
```

TD : Devinez à quel nombre je pense…

```
Algo nombreCacheCache
Constante NB_MYSTERE : entier <- aléa(1,100) #initialisation aléatoire
Variable saisie : entier
Début
    saisie <- saisir("A quel nombre entre 1 et 100 je pense ? ")
    TantQue saisie ≠ NB_MYSTERE
        Si saisie > NB_MYSTERE Alors
            écrire("C'est moins !")
        Sinon
            écrire("C'est plus !")
        FSi
        saisie <- saisir()
    FTq
    écrire("Bravo ! Vous avez trouvé !")
Fin
```

Les instructions itératives

TD : Que fait cet algorithme ?

```
Algo Piege
Variable nbTentatives : entier <- 1
Variable saisie : texte
Constante MAX_TENTATIVES : entier <- 5
Début
    saisie <- saisir("Quelle est la capitale de la France ? ")
    TantQue saisie ≠ "Paris" ou MAX_TENTATIVES - nbTentatives ≠ 0
        écrire("Mauvaise réponse !")
        écrire("Plus que " & MAX_TENTATIVES - nbTentatives & " tentative(s)")
        saisie <- saisir("Quelle est la capitale de la France ? ")
    FTq
    Si MAX_TENTATIVES - nbTentatives ≠ 0 Alors
        écrire("Bravo !")
    Sinon
        écrire("Revoyez votre géographie !")
    FSi
Fin
```

TD : Que fait cet algorithme ? Une première erreur corrigée

```
Algo UneErreurDeMoins
Variable nbTentatives : entier <- 1
Variable saisie : texte
Constante MAX_TENTATIVES : entier <- 5
Début
    saisie <- saisir("Quelle est la capitale de la France ? ")
    TantQue saisie ≠ "Paris" ou MAX_TENTATIVES - nbTentatives ≠ 0
        écrire("Mauvaise réponse !")
        écrire("Plus que " & MAX_TENTATIVES - nbTentatives & " tentative(s)")
        saisie <- saisir("Quelle est la capitale de la France ? ")
        nbTentatives <- nbTentatives + 1
    FTq
    Si MAX_TENTATIVES - nbTentatives ≠ 0 Alors
        écrire("Bravo !")
    Sinon
        écrire("Revoyez votre géographie !")
    FSi
Fin
```

TD : Que fait cet algorithme ? Une seconde erreur corrigée

```
Algo DeuxErreursDeMoins
Variable nbTentatives : entier <- 1
Variable saisie : texte
Constante MAX_TENTATIVES : entier <- 5
Début
    saisie <- saisir("Quelle est la capitale de la France ? ")
    TantQue saisie ≠ "Paris" et MAX_TENTATIVES - nbTentatives ≠ 0
        écrire("Mauvaise réponse !")
        écrire("Plus que " & MAX_TENTATIVES - nbTentatives & " tentative(s)")
        saisie <- saisir("Quelle est la capitale de la France ? ")
        nbTentatives <- nbTentatives + 1
    FTq
    Si MAX_TENTATIVES - nbTentatives ≠ 0 Alors
        écrire("Bravo !")
    Sinon
        écrire("Revoyez votre géographie !")
    FSi
Fin
```

TD : Que fait cet algorithme ?
Une troisième erreur corrigée

```
Algo TroisErreursDeMoins
Variable nbTentatives : entier <- 1
Variable saisie : texte
Constante MAX_TENTATIVES : entier <- 5
Début
    saisie <- saisir("Quelle est la capitale de la France ? ")
    TantQue saisie ≠ "Paris" et MAX_TENTATIVES - nbTentatives ≠ 0
        écrire("Mauvaise réponse !")
        écrire("Plus que " & MAX_TENTATIVES - nbTentatives & " tentative(s)")
        saisie <- saisir("Quelle est la capitale de la France ? ")
        nbTentatives <- nbTentatives + 1
    FTq
    Si saisie = "Paris" Alors
        écrire("Bravo !")
    Sinon
        écrire("Revoyez votre géographie !")
    FSi
Fin
```

Les instructions itératives

La boucle Répéter

Répéter

(ré)affectation de la variable de condition

instructions

TantQue condition **FRépéter**

- Exécute au moins une fois la boucle
  - Exemple :

```
Répéter
        nb <- saisir("Un nombre pair, svp")
TantQue nb mod 2 ≠ 0 FRépéter
```

Les instructions itératives

TD : Affichage de répliques de films

- Le programme affiche un menu avec une liste de films et une option pour quitter
  - Si l'utilisateur choisit l'un des films
    - le programme affiche une réplique de ce film
    - l'utilisateur peut choisir un autre film ou quitter
  - Si l'utilisateur choisit de quitter
    - Le programme affiche un message avant de se terminer

TD : Affichage de répliques de films

- Affichages et *saisies*

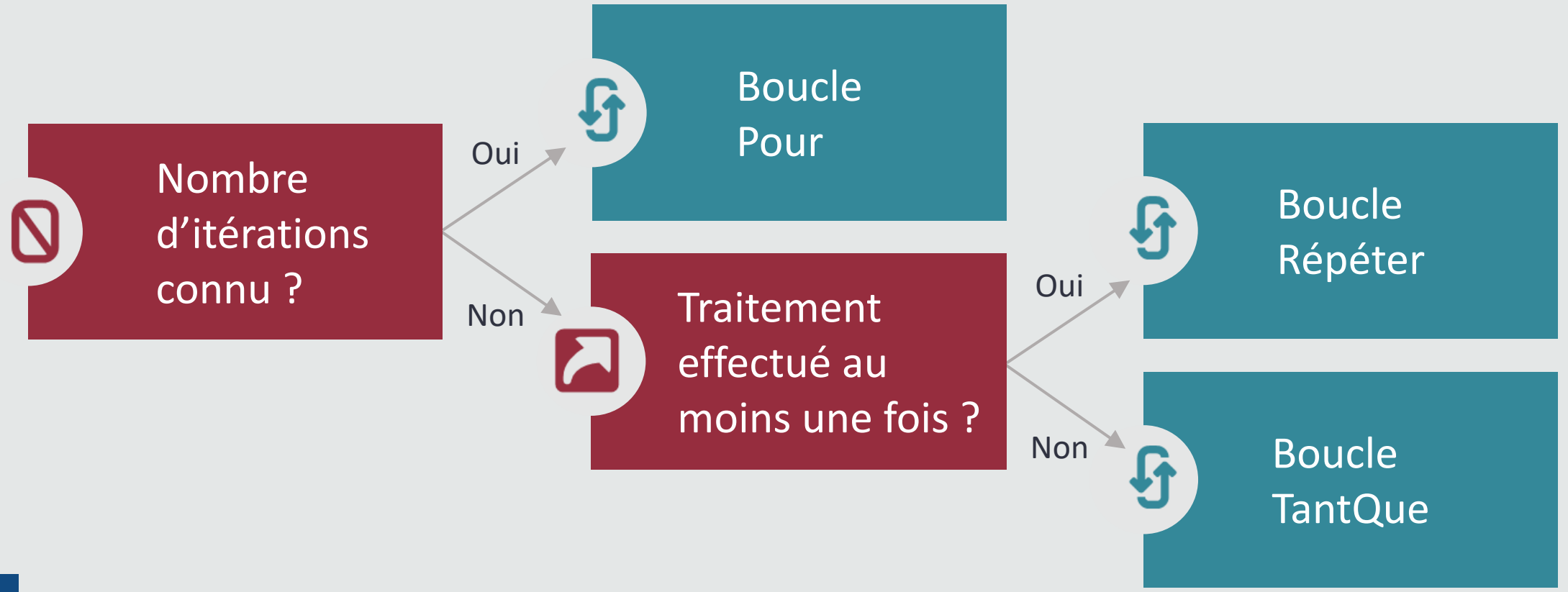
1 – Une réplique de la cité de la peur
2 – Une réplique de James Bond
3 – Une réplique de la vie est un long fleuve tranquille
4 – Une réplique de Star Wars
5 – Quitter cette magnifique application !
2
Je m'appelle Bond, James Bond
1
Attention, c'est une véritable boucherie !
5
Au revoir !

TD : Affichage de répliques de films

```
Algo repliquesDeFilms
Variable saisie : entier
Début
    écrire("1–Une réplique de la cité de la peur")
    écrire("2–Une réplique de James Bond")
    écrire("3–Une réplique de la vie est un long fleuve tranquille")
    écrire("4–Une réplique de Star Wars")
    écrire("5–Quitter cette magnifique application !")
    Répéter
        saisie <- saisir()
        Selon saisie
            cas 1 : écrire("Attention, c'est une véritable boucherie !")
            cas 2 : écrire("Je m'appelle Bond, James Bond !")
            cas 3 : écrire("Le lundi, c'est ravioli")
            cas 4 : écrire("La force soit avec toi, jeune padawan !")
            cas 5 : écrire("Au revoir !")
            autre : écrire("Saisie incorrecte")
        FSelon
    TantQue saisie ≠ 5 FRépéter
Fin
```

Les instructions itératives

Quelle boucle choisir ?



TD : À moi de trouver

- L'ordinateur doit deviner un nombre choisi par l'utilisateur
  - L'utilisateur lui indique si le nombre est plus grand (+), plus petit (-) ou s'il a trouvé le nombre (=)
  - Affichages et *saisies*

Choisissez un nombre compris entre 1 et 100, puis appuyez sur une touche
a
Je tente 45, est-ce plus, moins ou est-ce le nombre (+/-/=) ?
-
Je tente 10, est-ce plus, moins ou est-ce le nombre (+/-/=) ?
+
Je tente 22, est-ce plus, moins ou est-ce le nombre (+/-/=) ?
+
Je tente 27, est-ce plus, moins ou est-ce le nombre (+/-/=) ?
=
Super ! J'ai trouvé en 4 tentatives

Les instructions itératives

TD : À moi de trouver

```
Algo devineUnNombre
Variable min : entier <- 1                Variable max : entier <- 100          Variable reponse : caractère
Variable nbTentatives : entier <- 0       Variable valeur : entier              Variable saisieOk : booléen
Début  # le caractère saisi n'a pas d'importance. C'est pour attendre que l'utilisateur ai choisi son nombre
    saisir("Choisissez un nombre compris entre " & min & " et " & max & ", puis appuyez sur une touche")
    Répéter
        valeur <- aléa(min, max)
        écrire("Je tente " & valeur & ", est-ce plus, moins ou est-ce le nombre (+/-/=) ?")
        nbTentatives <- nbTentatives + 1
        Répéter
            saisieOk <- VRAI
            reponse <- saisir()
            Selon reponse
                cas '+' : min <- valeur
                cas '-' : max <- valeur
                cas '=' : écrire("Super ! J'ai réussi en " & nbTentatives & "tentatives")
                autre : saisieOk <- FAUX
                        écrire("Erreur de saisie. Saisissez + - ou =")
            FSelon
        TantQue non saisieOk FRépéter
    TantQue reponse ≠ '=' FRépéter
Fin
```

TD : Saisie d'un multiple de 3

- Affichages et *saisies*

Entrez un multiple de 3
7579
Erreur 7579 n'est pas un multiple de 3
16427
Erreur 16427 n'est pas un multiple de 3
51321
Ok : 51321 est un multiple de 3

Les instructions itératives

TD : Saisie d'un multiple de 3

```
Algo multiple
Variable saisie : entier
Constante MULTIPLE : entier <- 3
Début
    écrire("Entrer un multiple de " & MULTIPLE)
    saisie <- saisir()
    TantQue saisie mod MULTIPLE ≠ 0
        écrire("Erreur " & saisie & " n'est pas un multiple de " & MULTIPLE)
        saisie <- saisir()
    FTq
    écrire("Ok " & saisie & " est un multiple de " & MULTIPLE)
Fin
```

Les instructions itératives

TD : Rendez la monnaie !

- Programmer le monnayeur de la machine à café
  - Prix du café : 0,60 €
  - Pièces acceptées : 2 € 1 € 0,50 € 0,20 € 0,10 € 0,05 €
  - Affichages et *saisies*

Entrez la valeur de la pièce :
2
Voici votre café et votre monnaie (1,40€) :
1 pièce(s) de 1€
2 pièce(s) de 0,20€

- Autres affichages et *saisies* possibles

Entrez la valeur de la pièce :
0,20
Crédit insuffisant (0,20€/0,60€)
0,01
Pièce non acceptée, entrez une autre pièce
0,20
Crédit insuffisant (0,40€/0,60€)
0,50
Voici votre café et votre monnaie (0,30 €) :
1 pièce(s) de 0,20€
1 pièce(s) de 0,10€

Les instructions itératives

TD : Rendez la monnaie !

```
Algo monnaieCafe
# Récupère des pièces jusqu'à ce que le crédit soit suffisant puis rend la monnaie
Variable credit : réel <- 0                Variable nbPieces : entier
Variable saisie : réel                     Constante PRIX_CAFE : réel <- 0,60
Début
    écrire("Entrez la valeur de la pièce : ")
    Répéter
        saisie <- saisir()
        # Si la pièce est acceptable
        Selon
        cas 2 ; 1 ; 0,5 ; 0,2 ; 0,1 ; 0,05 :
            credit <- credit + saisie
            Si credit < PRIX_CAFE Alors
                écrire("Crédit insuffisant (" & credit & "€/" & PRIX_CAFE & "€)")
            FSi
        autre :
            écrire("Pièce non acceptée, entrez une autre pièce")
        FSelon
    TantQue credit < PRIX_CAFE FRépéter
    credit <- credit - PRIX_CAFE
    écrire("Voici votre café et votre monnaie (" & credit & "€) :")
```

Les instructions itératives

TD : Rendez la monnaie !

```
# Rendu de la monnaie en pièces de 1€
nbPieces <- 0
TantQue credit ≥ 1
    nbPieces <- nbPieces + 1
    credit <- credit - 1
FTq
Si nbPieces > 0 Alors
    écrire(nbPieces & "pièce(s) de 1€")
FSi
# Rendu de la monnaie en pièces de 0,50€
nbPieces <- 0
TantQue credit ≥ 0,5
    nbPieces <- nbPieces + 1
    credit <- credit – 0,5
FTq
Si nbPieces > 0 Alors
    écrire(nbPieces & "pièce(s) de 0,50€")
FSi
```

TD : Rendez la monnaie !

```
# Rendu de la monnaie en pièces de 0,20€
nbPieces <- 0
TantQue credit ≥ 0,2
    nbPieces <- nbPieces + 1
    credit <- credit – 0,2
FTq
Si nbPieces > 0 Alors
    écrire(nbPieces & "pièce(s) de 0,20€")
FSi
# Rendu de la monnaie en pièces de 0,10€
nbPieces <- 0
TantQue credit ≥ 0,1
    nbPieces <- nbPieces + 1
    credit <- credit – 0,1
FTq
Si nbPieces > 0 Alors
    écrire(nbPieces & "pièce(s) de 0,10€")
FSi
```

Les instructions itératives

TD : Rendez la monnaie !

```
# Rendu de la monnaie en pièces de 0,05€
nbPieces <- 0
TantQue credit ≥ 0,05
    nbPieces <- nbPieces + 1
    credit <- credit – 0,05
FTq
Si nbPieces > 0 Alors
    écrire(nbPieces & "pièce(s) de 0,05€")
FSi
Fin
```

Les instructions itératives

TD : ASCII Art !

- Dessiner un rectangle

```
Largeur ?
8
Hauteur?
5
Caractère?
#
########
########
########
########
########
```

écrireSRC()
permet d'écrire sans retourner à la ligne

Les instructions itératives

TD : ASCII Art !

```
Algo dessin
# Affiche un rectangle rempli avec un même caractère
Variable largeur, hauteur, i, j : entier
Variable car : caractère
Début
    largeur <- saisir("Largeur ? ")
    hauteur <- saisir("Hauteur ? ")
    car <- saisir("Caractère ? ")
    Pour j <- 1 à hauteur
        Pour i <- 1 à largeur
            écrireSRC(car)
        FPour
        écrire()
    FPour
Fin
```

```
Largeur ?
8
Hauteur?
5
Caractère?
#
########
########
########
########
########
```

TD : Encore des exercices…

- Dessiner une des formes suivantes selon le choix de l'utilisateur

```
taille ?
5
Caractère ?
#
forme ?
1 – rectangle plein
2 – rectangle creux
3 – croix de Saint-André
4 – triangle
5 – losange
6 – damier
2
#####
#   #
#   #
#   #
#####
```

```
#####
#####
#####
#####
#####
```
Rectangle plein

```
#
##
# #
#  #
#####
```
Triangle

```
#   #
 # #
  #
 # #
#   #
```
Croix de Saint-André

```
  #
 # #
#   #
 # #
  #
```
Losange

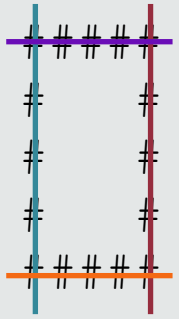
```
# # #
 # #
# # #
 # #
# # #
```
Damier

Remarque : le nombre de lignes est égal au nombre de colonnes

TD : Encore des exercices…

Numéro de ligne = 1

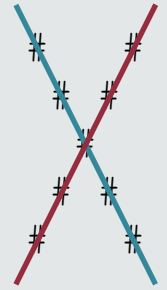
Numéro de ligne
= Taille



Numéro de colonne = 1

Numéro de colonne
= Taille

Numéro de colonne = Numéro de ligne



Numéro de colonne + Numéro de ligne
= Taille + 1

TD : Encore des exercices…

- Rendez la monnaie (version 2)
  - Gérer un stock de pièces
    - Initialement vous avez un stock de pièces
      - Par exemple

2 €	1 €	0€50	0€20	0€10	0€05
10	3	0	10	10	5

    - Modifier ce stock en fonction des pièces insérées et des pièces distribuées

Algorithmique

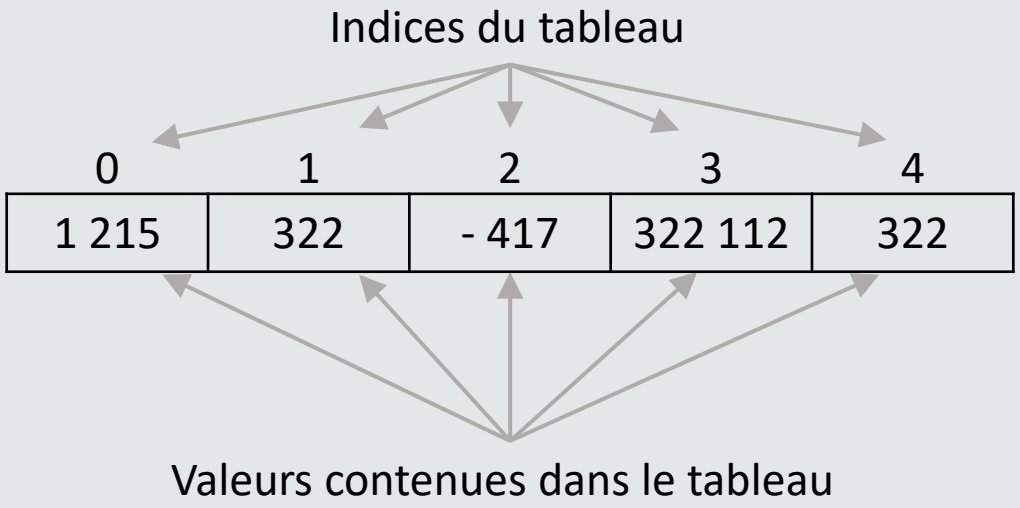
Module 5 - Les tableaux

Objectifs

- Comprendre le fonctionnement des tableaux
- Être capable de manipuler un tableau
- Comprendre le fonctionnement des tableaux à plusieurs dimensions (multidimensionnels)
- Être capable de manipuler des tableaux à plusieurs dimensions

Les tableaux

- Un tableau est un ensemble ordonné de valeurs d'un type de données
- Exemple



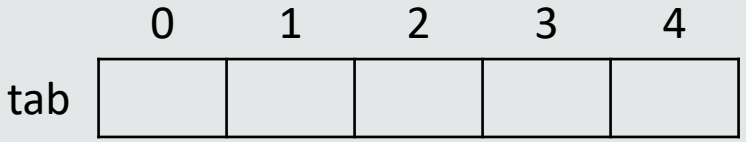
Déclaration d'un tableau

```
Variable  monTableau  :  type[TAILLE]
```

- Mot-clé indiquant la déclaration
- Nom du tableau
- Type des données contenues dans le tableau
- Taille du tableau

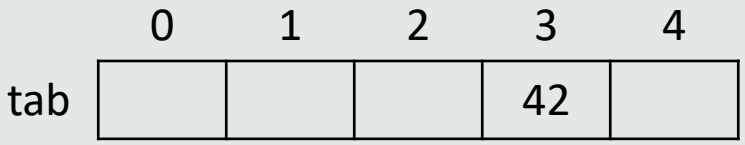
- Exemple

```
Variable  tab  :  entier[5]
```



Accès à un élément d'un tableau

nomTableau[indice]



Nom du tableau

Les crochets indiquent l'accès à un élément du tableau

Indice de l'élément dans le tableau

- Exemple accès en écriture

```
tab[3] <- 42      # affectation de la valeur 42 dans la case n°3 du tableau tab
```

- Exemple accès en lecture

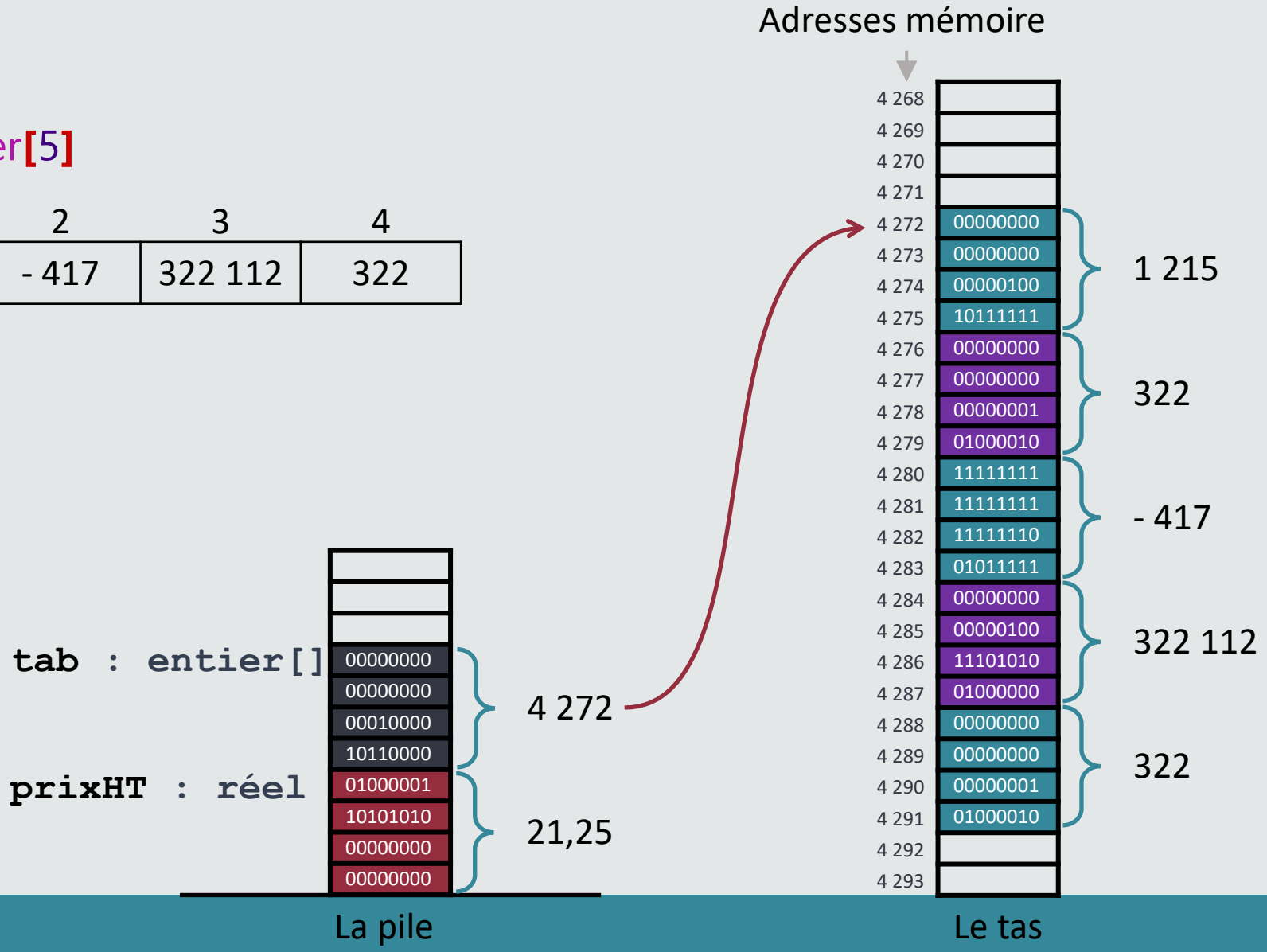
```
écrire(tab[3])    # affichage de la valeur contenue dans la case n°3 de tab
```

Les tableaux

En mémoire...

Variable tab : entier[5]

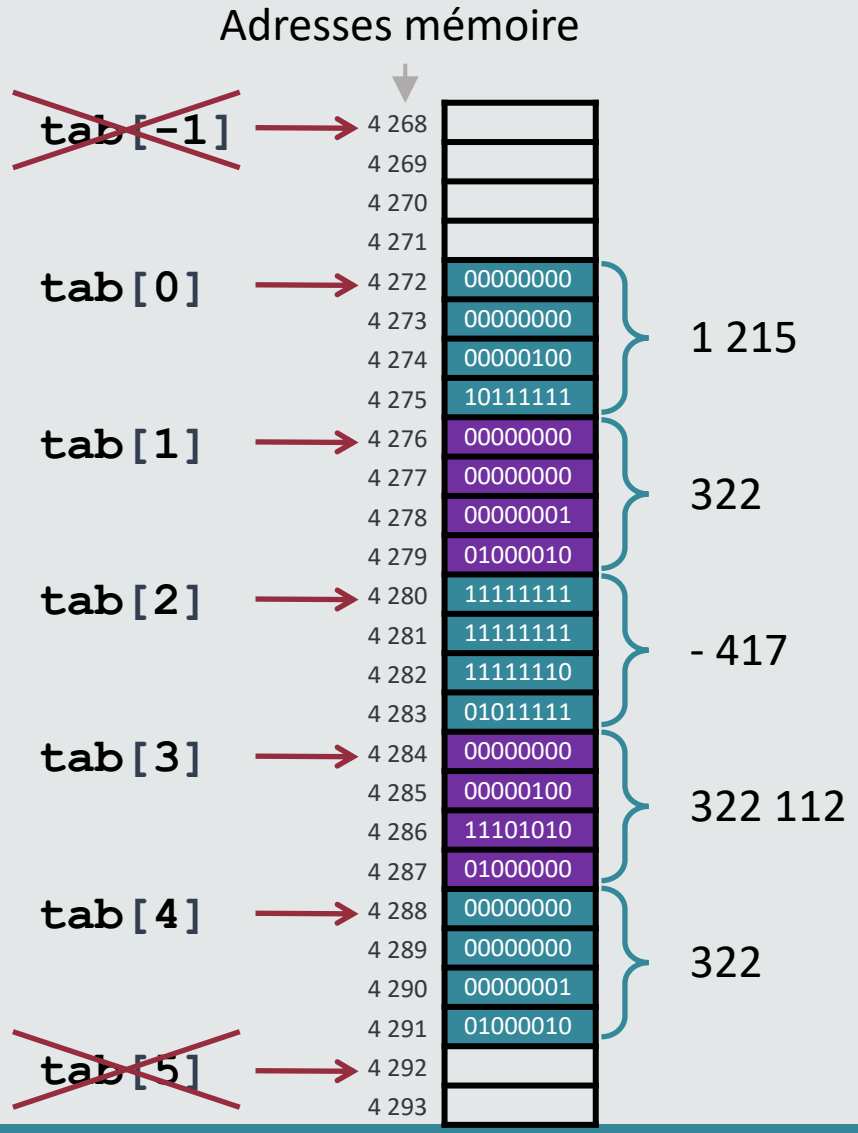
0	1	2	3	4
1 215	322	- 417	322 112	322



Accès à un élément

Adresse de tab[i] =
Adresse du tableau + (i × taille d'un élément du tableau)

Attention : la valeur de l'indice doit être comprise entre 0 et la taille du tableau - 1



TD : Nombre d'occurrences

- Saisir des chiffres compris entre 0 et 9
- -1 pour terminer la saisie
- Calculer le nombre de fois où l'utilisateur a saisi chaque chiffre

TD : Nombre d'occurrences

Entrer une valeur comprise entre 0 et 9 :
2
Autre valeur, svp :
9
Autre valeur, svp :
2
Autre valeur, svp :
2
Autre valeur, svp :
0
Autre valeur, svp :
-1
Nombre de 0 : 1
Nombre de 1 : 0
Nombre de 2 : 3
…
Nombre de 9 : 1

	0	1	2	3	4	5	6	7	8	9
tab	1	0	3	0	0	0	0	0	0	1

TD : Nombre d'occurrences

```
Algo nbOcc                                  # Demande des chiffres et calcule le nombre d'occurrences de chacun
Constante TAILLE : entier <- 10
Variable tab : entier[TAILLE]               Variable i : entier
Début
    Pour i <- 0 à TAILLE-1                  # Initialisation du tableau
        tab[i] <- 0
    Fpour
...
```

TD : Nombre d'occurrences

```
Algo nbOcc                                  # Demande des chiffres et calcule le nombre d'occurrences de chacun
Constante TAILLE : entier <- 10             Constante STOP : entier <- -1
Variable tab : entier[TAILLE]               Variable i, valeur : entier
Début
    Pour i <- 0 à TAILLE-1                  # Initialisation du tableau
        tab[i] <- 0
    Fpour                                   # Saisie des valeurs
    valeur <- saisir("Entrer une valeur comprise entre 0 et " & TAILLE-1 & " : ")
    TantQue valeur ≠ STOP
        Si valeur < 0 Ou valeur >= TAILLE Alors
            écrire("N'est pas une valeur entre 0 et " & TAILLE-1 & " !")
        Sinon
            tab[valeur] <- tab[valeur] + 1
        FSi
        valeur <- saisir("Autre valeur, svp : ")
    FTq
...
```

TD : Nombre d'occurrences

```
Algo nbOcc                                  # Demande des chiffres et calcule le nombre d'occurrences de chacun
Constante TAILLE : entier <- 10             Constante STOP : entier <- -1
Variable tab : entier[TAILLE]               Variable i, valeur : entier
Début
    Pour i <- 0 à TAILLE-1                  # Initialisation du tableau
        tab[i] <- 0
    Fpour                                   # Saisie des valeurs
    valeur <- saisir("Entrer une valeur comprise entre 0 et " & TAILLE-1 & " : ")
    TantQue valeur ≠ STOP
        Si valeur < 0 Ou valeur >= TAILLE Alors
            écrire("N'est pas une valeur entre 0 et " & TAILLE-1 & " !")
        Sinon
            tab[valeur] <- tab[valeur] + 1
        FSi
        valeur <- saisir("Autre valeur, svp : ")
    FTq
    Pour i <- 0 à TAILLE-1                  # Affichage des résultats
        écrire("Nombre de " & i & " : " & tab[i])
    FPour
Fin
```

Les tableaux

TD : Palindrome

- Un palindrome est un mot dont l'ordre des lettres reste le même qu'il soit lu de gauche à droite ou de droite à gauche
- Exemples : Kayak, ici, Anna, radar, rotor…
- Écrire un programme qui :
  - demande à l'utilisateur de saisir un mot sans accent, en minuscules et suivi de #
  - indique à l'utilisateur si ce mot est un palindrome

TD : Palindrome

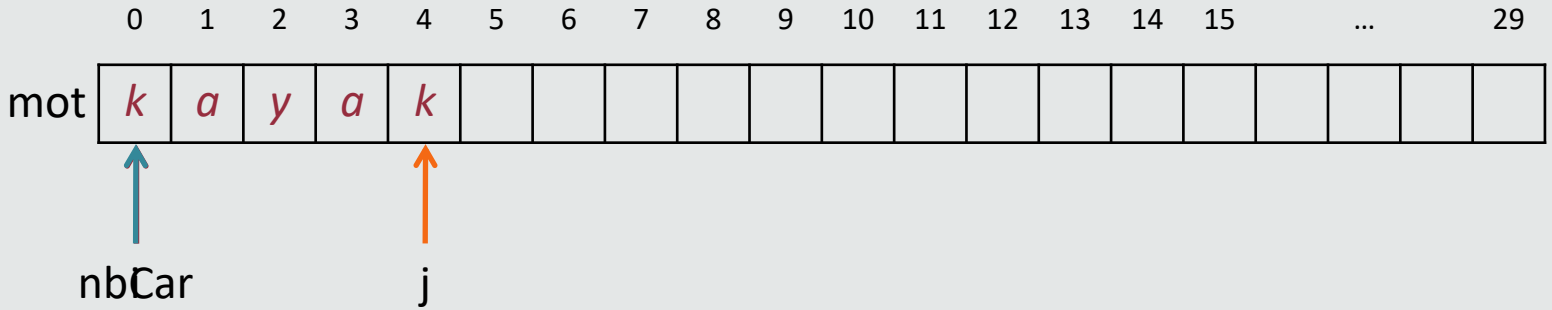
- Exemple d'affichages et de saisies

Entrez un mot puis tapez #
kayak#
kayak est un palindrome

TD : Palindrome

- Le mot est saisi caractère par caractère
- Chaque caractère saisi est stocké dans une case d'un tableau
- Une fois la saisie terminée, le tableau est parcouru en partant simultanément du premier et du dernier caractère

Entrez un mot puis tapez #
kayak#
kayak est un palindrome



Les tableaux

TD : Palindrome

```
Algo palindrome # Demande la saisie d'un mot et indique s'il s'agit d'un palindrome
Constante TAILLE : entier <- 30          Constante STOP : caractère <- '#'
Variable mot : caractère[TAILLE]         Variable nbCar : entier <- 0
Variable palin : booléen <- Vrai         Variable i : entier
Début
 #### Saisie du mot ####
 écrire("Entrez un mot puis tapez ", STOP)
 Répéter
  mot[nbCar] <- saisir()
  Si mot[nbCar] ≠ STOP Alors
   Si mot[nbCar] < 'a' ou mot[nbCar] > 'z' Alors
    écrire("Ce caractère n'est pas autorisé, il ne faut écrire que des lettres minuscules sans accent ni cédille")
    Pour i <- 0 à nbCar-1        # Réécriture du début du mot
     écrireSRC(mot[i])
    FPour
   Sinon
    nbCar <- nbCar + 1
   FSi
  FSi
 TantQue mot[nbCar] ≠ STOP FRépéter
```

Les tableaux

TD : Palindrome

```
i <- 0
#### Test du mot pour voir si c'est un palindrome ####
TantQue palin et i<nbCar div 2
  palin <- mot[i]=mot[nbCar-1-i]
  i <- i+1
FTq
écrireSRC("Le mot ")
Pour i <- 0 à nbCar-1
  écrireSRC(mot[i])
FPour
Si palin Alors
  écrire(" est un palindrome")
Sinon
  écrire(" n'est pas un palindrome")
FSi
Fin
```

TD : Moyenne de notes (version 4)

- Saisir des notes et les stocker dans un tableau
- -1 pour terminer la saisie

Note ?
12
Note ?
15
Note ?
8
Note ?
7
Note ?
-1
La moyenne des notes (12; 15; 8; 7) est de 10,5

TD : Moyenne de notes (version 4)

```
Algo MoyenneDesNotesV4
# Calcule la moyenne des notes saisies par l'utilisateur (il saisit -1 pour finir la saisie).
# Les notes sont stockées dans un tableau.
Constante TAILLE    : entier <- 100
Constante STOP      : entier <- -1
Variable notes      : réel[TAILLE]
Variable nbVal      : entier <- 0
Variable saisie     : réel
Variable i          : entier
Variable total      : réel
Début
    saisie <- saisir("Note ? ")
    TantQue saisie ≠ STOP et nbVal < TAILLE
        notes[nbVal] <- saisie
        nbVal <- nbVal + 1
        saisie <- saisir("Note ? ")
    FTq
```

TD : Moyenne de notes (version 4)

```
# Si c'est l'utilisateur qui a clos la saisie
Si saisie = STOP Alors
    Si nbVal ≠ 0 Alors
        écrireSRC("La moyenne des notes (" & note[0])
        total <- notes[0]
        Pour i <- 1 à nbVal-1
            total <- total + notes[i]
            écrireSRC("; " & note[i])
        FPour
        écrire(") est de " & total/nbVal)
    Sinon
        écrire("Aucune note n'a été saisie")
    FSi
Sinon
    écrire("La capacité du tableau a été dépassée, désolé")
FSi
Fin
```

Déclaration d'un tableau à deux dimensions

```
Variable tableau : type[DIM1][DIM2]
```

Mot-clé indiquant la déclaration

Nom du tableau

Type des données contenues dans le tableau

Taille du tableau

- Exemple

```
Variable tableau : entier[3][10]
```

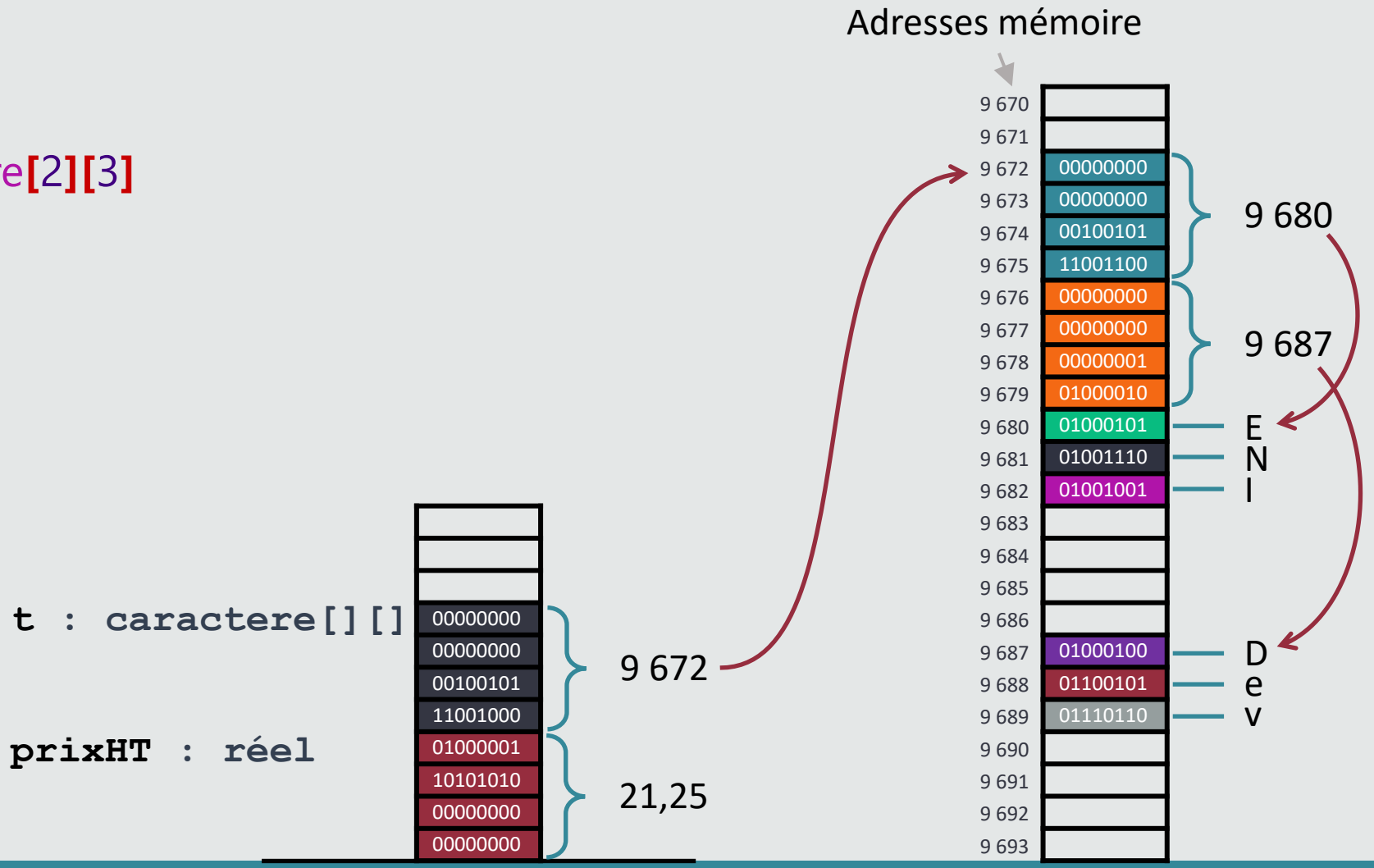
		0	1	2	3	4	5	6	7	8	9
tableau	0										
	1										
	2										

Les tableaux

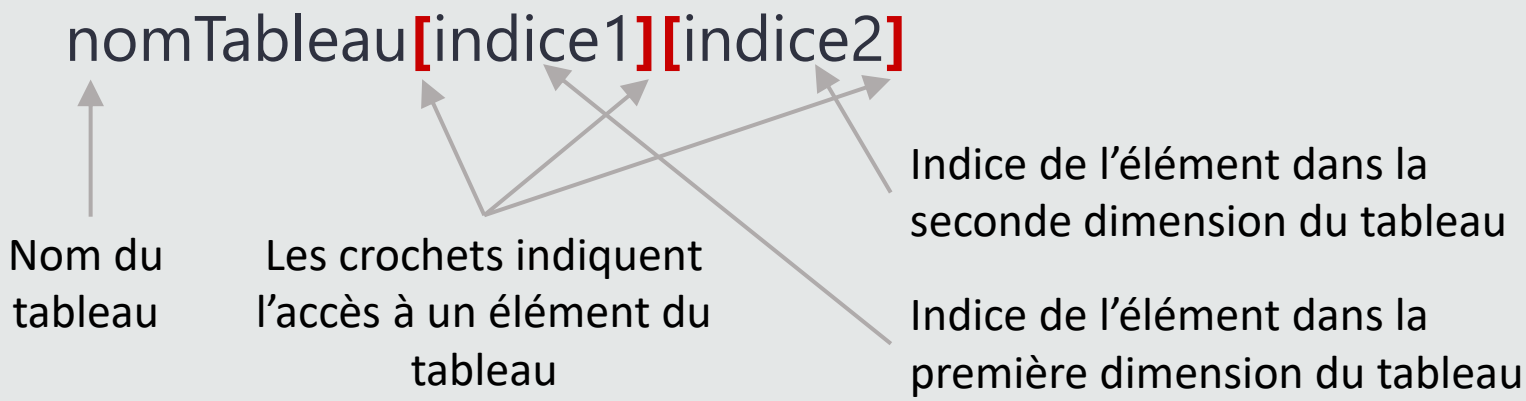
En mémoire...

Variable t : caractère[2][3]

E	N	I
D	e	v



Accès à un élément d'un tableau à 2 dimensions



- Exemple d'accès en écriture

```
tab[2][8] <- 4 # affectation de la valeur 4 dans la case de coordonnée (2,8) de tab
```

- Exemple d'accès en lecture

```
écrire(tab[3][0]) # affichage de la valeur contenue dans la case de coordonnées (3,0) de tab
```

TD : Que fait-il donc ?

```
Algo QueFaitIlDonc
# Numérote toutes les cases du tableau de 1 à TAILLE×TAILLE colonne par colonne
Constante TAILLE     : entier <- 3
Variable i, j, val : entier
Variable tab2d : entier[TAILLE][TAILLE]
Début
    val <- 1
    Pour j <- 0 à TAILLE - 1
        Pour i <- 0 à TAILLE - 1
            tab2d[j][i] <- val
            val <- val + 1
        FPour
    FPour
Fin
```

1	2	3
4	5	6
7	8	9

TD : Que fait-il donc ?

- Modifier l'algorithme précédent pour mettre les valeurs suivantes dans le tableau

```
Algo QueFaitIlDoncV2
# Numérote toutes les cases du tableau de 1 à TAILLE×TAILLE
# colonne par colonne
Constante TAILLE     : entier <- 4
Variable i, j, val : entier
Variable tab2d : entier[TAILLE][TAILLE]
Début
        val <- 1
        Pour j <- 0 à TAILLE - 1
                Pour i <- 0 à TAILLE - 1
                        tab2d[i][j] <- val
                        val <- val + 1
                FPour
        FPour
Fin
```

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

TD : Que fait-il donc ?

- Puis pour mettre les valeurs suivantes

```
Algo QueFaitIlDoncV3
# Numérote toutes les cases du tableau indiquant la distance par
# rapport à la première case
Constante TAILLE     : entier <- 4
Variable i, j : entier
Variable tab2d : entier[TAILLE][TAILLE]
Début
        Pour j <- 0 à TAILLE - 1
                Pour i <- 0 à TAILLE - 1
                        tab2d[j][i] <- i+ j
                FPour
        FPour
Fin
```

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

Les tableaux

TD : Matrix

- Créer un tableau de taille 20×30 dans lequel seront ajoutés des caractères choisis aléatoirement

- Afficher votre matrix

aléa() permet également de tirer aléatoirement un caractère
Exemple : aléa(' ', '2')

TD : Micro bataille navale

- Plateau de jeu de 4×4 cases
- Une des cases du tableau (choisie aléatoirement) contient un bateau
- Affichages et *saisies*

```
????
????
????
????
Quelle colonne ? 2
Quelle ligne ? 2
Plouf ! À l'eau !
????
?~??
????
????
Quelle colonne ? 4
Quelle ligne ? 1
Plouf ! À l'eau !
???~
?~??
????
????
Quelle colonne ? 1
Quelle ligne ? 3
Boom ! Touché coulé
Bravo, vous avez gagné !
```

Les tableaux

TD : Micro bataille navale

```
Algo microBatailleNavale
# Jeu de la Bataille Navale avec un seul bateau d'une case
Constante HAUTEUR     : entier <- 4
Constante LARGEUR     : entier <- 4

Variable  plateau     : caractère[HAUTEUR][LARGEUR]
Constante PLOUF       : caractère <- '~'
Constante BATEAU      : caractère <- 'b'
Constante EAU         : caractère <- 'e'
Variable  i, j        : entier
Début
    # Initialisation du plateau de jeu
    Pour j <- 0 à HAUTEUR-1
        Pour i <- 0 à LARGEUR-1
            plateau[j][i] <- EAU
        FPour
    FPour
    # Dépôt d'un bateau d'une case sur une case choisie aléatoirement
    i <- aléa(0, HAUTEUR-1)
    j <- aléa(0, LARGEUR-1)
    plateau[i][j] <- BATEAU
```

Les tableaux

TD : Micro bataille navale

```
Répéter
    Pour j <- 0 à HAUTEUR-1          # Affichage du plateau de jeu
        Pour i <- 0 à LARGEUR-1
            Si plateau[j][i] = BATEAU ou plateau[j][i] = EAU Alors
                écrireSRC("?")
            Sinon
                écrireSRC(plateau[j][i])
            FSi
        FPour
        écrire()
    FPour
    Répéter                                  # Saisie des coordonnées de tir
        i <- saisir("Quelle colonne (entre 1 et "&LARGEUR&") : ") - 1
        j <- saisir("Quelle ligne (entre 1 et "&HAUTEUR&") : ") - 1
    TantQue i < 0 ou i ≥ LARGEUR ou j < 0 ou j ≥ HAUTEUR FRépéter
    Si plateau[j][i] ≠ BATEAU Alors          # test du tir
        plateau[j][i] <- PLOUF
        écrire("Plouf !  ")
    Fsi
TantQue plateau[j][i] ≠ BATEAU FRépéter
écrire("Touché Coulé ! Bravo, vous avez gagné !")
Fin
```

TD : Morpion

- Créer un jeu de morpion
  - Dans une grille de 3 cases par 3, les joueurs écrivent tour à tour leur symbole (X et O) dans une case vide
  - Le premier joueur qui parvient à aligner (en ligne, en colonne ou en diagonale) 3 de ses symboles a gagné

Algorithmique

Module 6 - Les procédures et fonctions

Objectifs

- Appréhender le paradigme de la programmation procédurale
- Savoir écrire des procédures et des fonctions et y faire appel

La notion de sous-algorithme

- Un algorithme peut faire appel à un sous-algorithme
- Le sous-algorithme prend alors la main, réalise ses instructions puis rend la main à l'algorithme qui l'a appelé

Retour sur la recette de cuisine

- Une recette de cuisine :
  - Liste des ingrédients

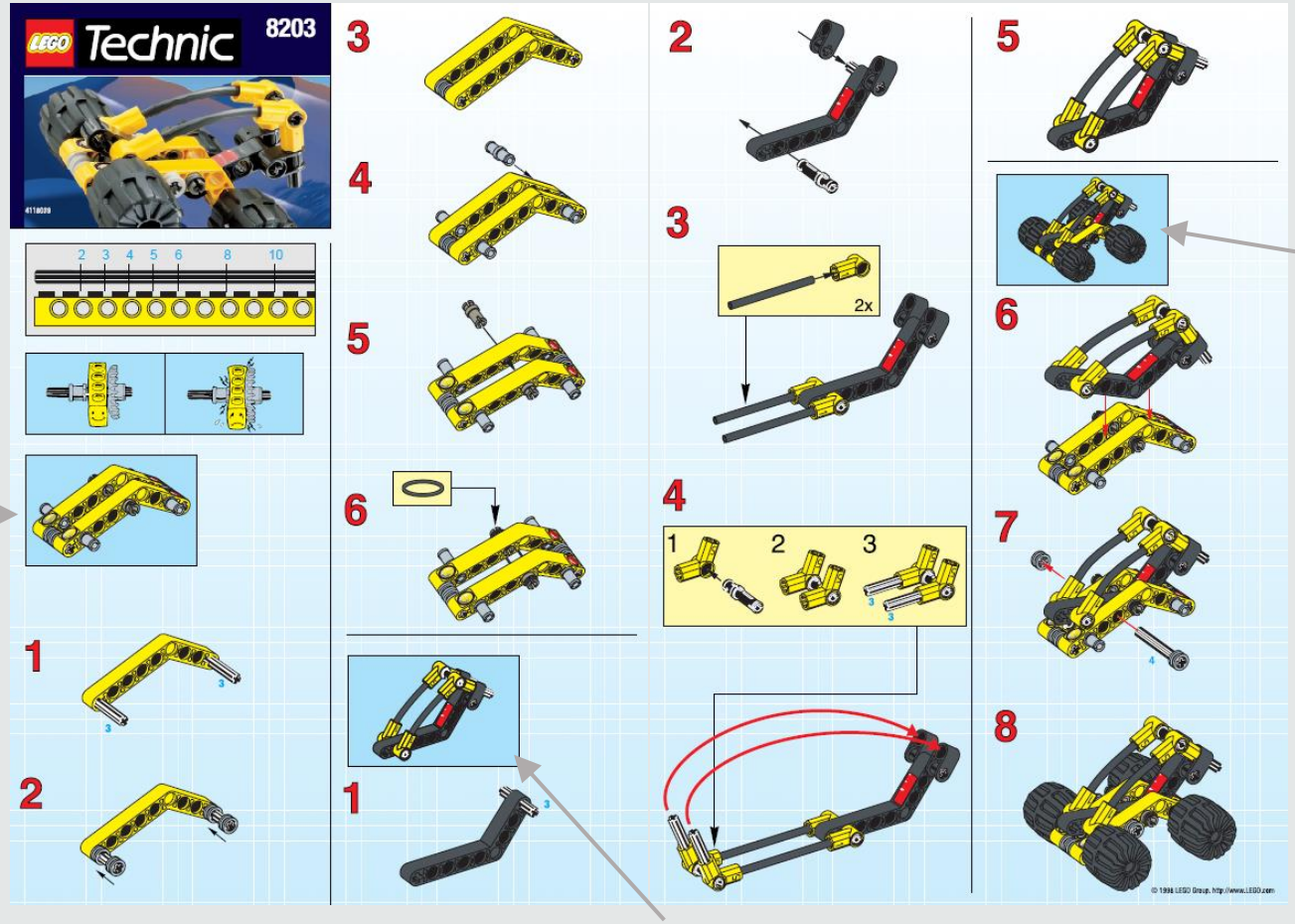
250 g de farine	1 pincée de sel	5 cl de rhum
4 œufs	50 grammes de beurre	
½ L de lait	1 sachet de sucre vanillé	

[www.pate-a-crepe.info]

  - Les étapes
    - Dans un saladier, verser la farine et les œufs.
    - Puis progressivement ajoutez le lait tout en mélangeant avec votre fouet.
    - Ajoutez le sucre vanillé, la pincée de sel.
    - Laisser reposer la pâte.
    - Versez une demi-louche de votre pâte à crêpe et faites cuire 1 à 2 minutes par face.
    - Conseil : servir avec du caramel au beurre salé (recette page suivante).

Appel à une autre recette

Retour sur la voiture en Lego



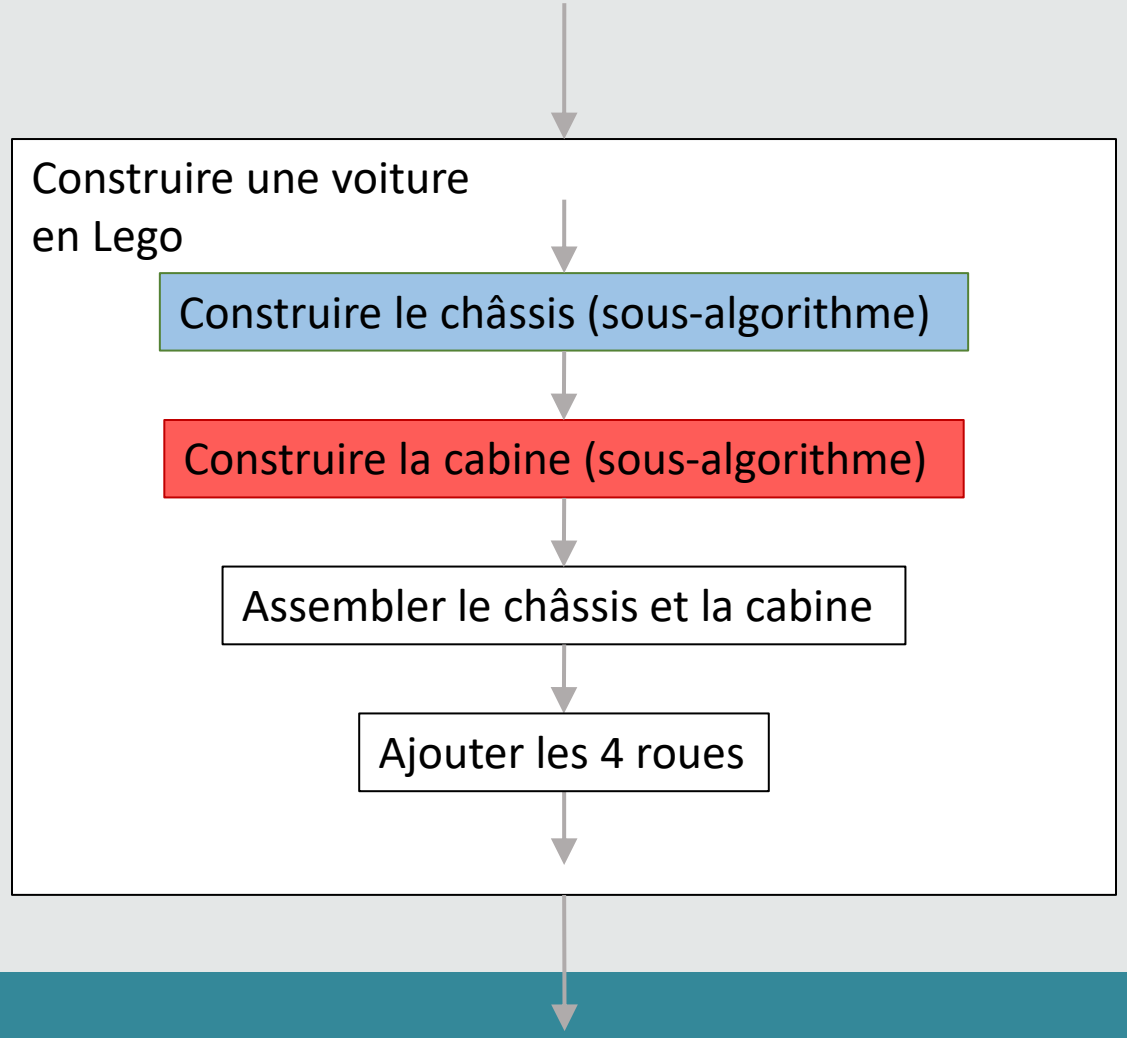
[www.lego.com]

À partir d'éléments Lego de base, un élément complexe est créé

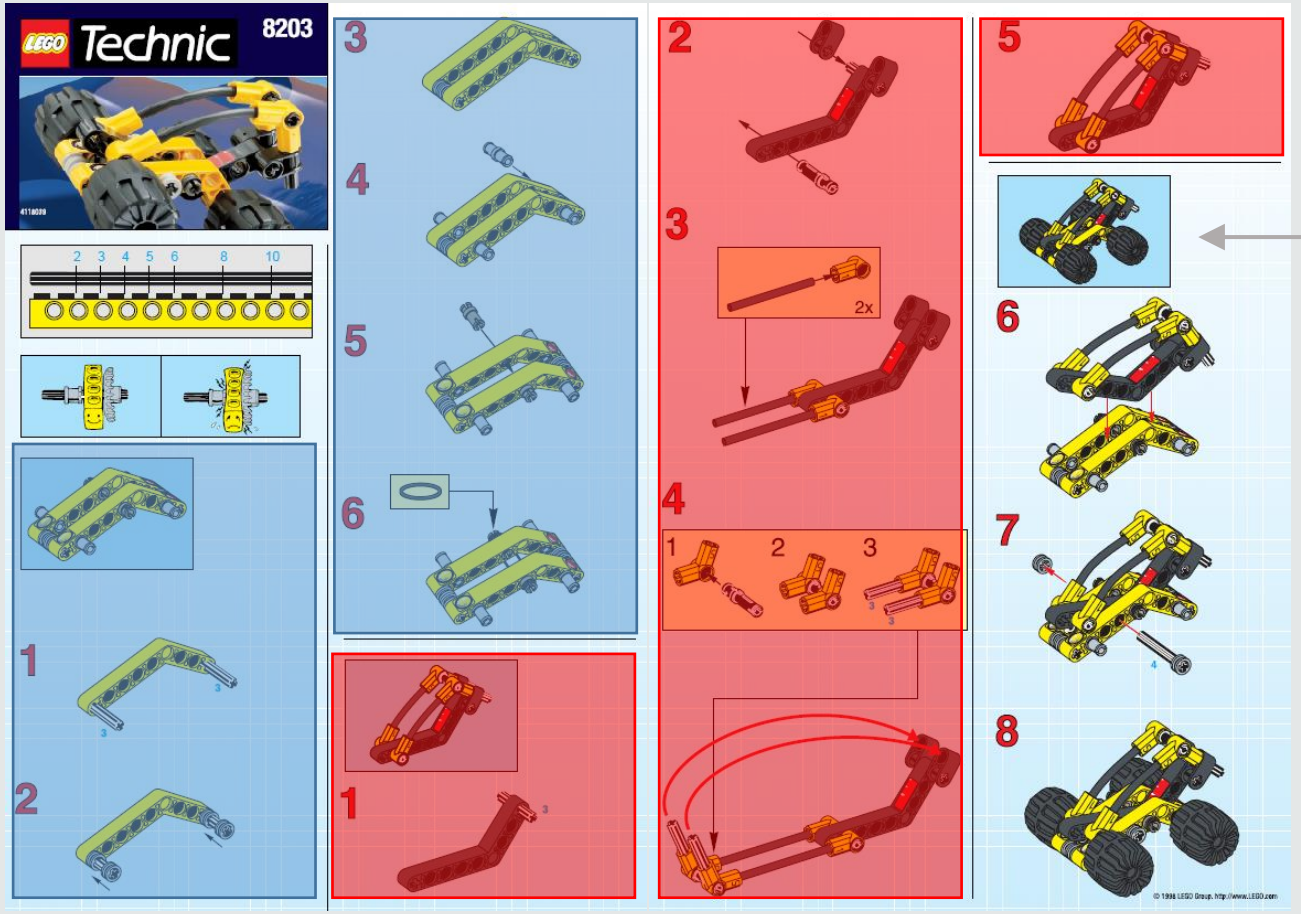
À partir d'éléments Lego de base et d'éléments complexes, un autre élément complexe est créé

À partir d'éléments Lego de base, un élément complexe est créé

Retour sur la voiture en Lego



Retour sur la voiture en Lego



[www.lego.com]

Algorithme Principal

Sous-algorithme creationChassis

Sous-algorithme creationCabine

Les procédures et fonctions

Intérêt des sous-algorithmes

- Évite la recopie du code
  - La recette du caramel est écrite une seule fois dans le livre de recette et pas à chaque fois qu'une recette utilise du caramel (Ile flottante, Gâteau à l'ananas, pièce montée…)

- Structure le programme
  - La notice Lego est séparée en 3 grandes étapes
    - Construction du châssis
    - Construction de la cabine
    - Assemblage général

Deux types de sous-programmes

- Les procédures
  - Un sous-programme constitué d'une suite d'instructions
  - Exemple : **`Procédure écrire`**

- Les fonctions
  - Un sous-programme constitué d'une suite d'instructions
  - À la fin de son exécution, retourne une valeur à l'algorithme appelant
  - Exemple : **`Fonction saisir`**, **`Fonction aléa`**

Les procédures et fonctions

Déclaration d'une procédure

```
Procédure nom(listeParametres)
Début
        instructions
Fin
```

- Exemple :

```
Procédure afficheNfois(t : texte, n : entier)
Variable i : entier
Début
        Pour i <- 1 à n
                écrire(t)
        FPour
Fin
```

Déclaration d'une fonction

```
Fonction nom(listeParametres) Retourne type
Début
        instructions
        Retourne valeur
Fin
```

- Exemple :

```
Fonction puissance(a : réel, n : entier) Retourne réel
Variable i : entier
Variable p : réel <- 1
Début
        Pour i <- 1 à n
                p <- p * a
        FPour
        Retourne p
Fin
```

Appel d'une procédure ou d'une fonction

- Procédure

```
afficheNfois("Bonjour !", 6)
```

- Fonction

```
Variable puiss : réel
...
puiss <- puissance(10, 6)
```

Exemple

```
Fonction puissance(a : réel, n : entier) Retourne réel
Variable i : entier
Variable p : réel <- 1
Début
    Pour i <- 1 à n
        p <- p * a
    FPour
    Retourne p
Fin # de la fonction puissance
Algo afficheLaPuissance
# Demande la saisie d'une valeur puis un exposant et affiche valeur^exposant
Variable  val, p : réel
Variable  exp    : entier
Début
    val <- saisir("Entrer une valeur")
    exp <- saisir("Entrer l'exposant")
    p <- puissance(val, exp)
    écrire(val & " puissance " & exp & " = " & p)
Fin
```

Les procédures et fonctions

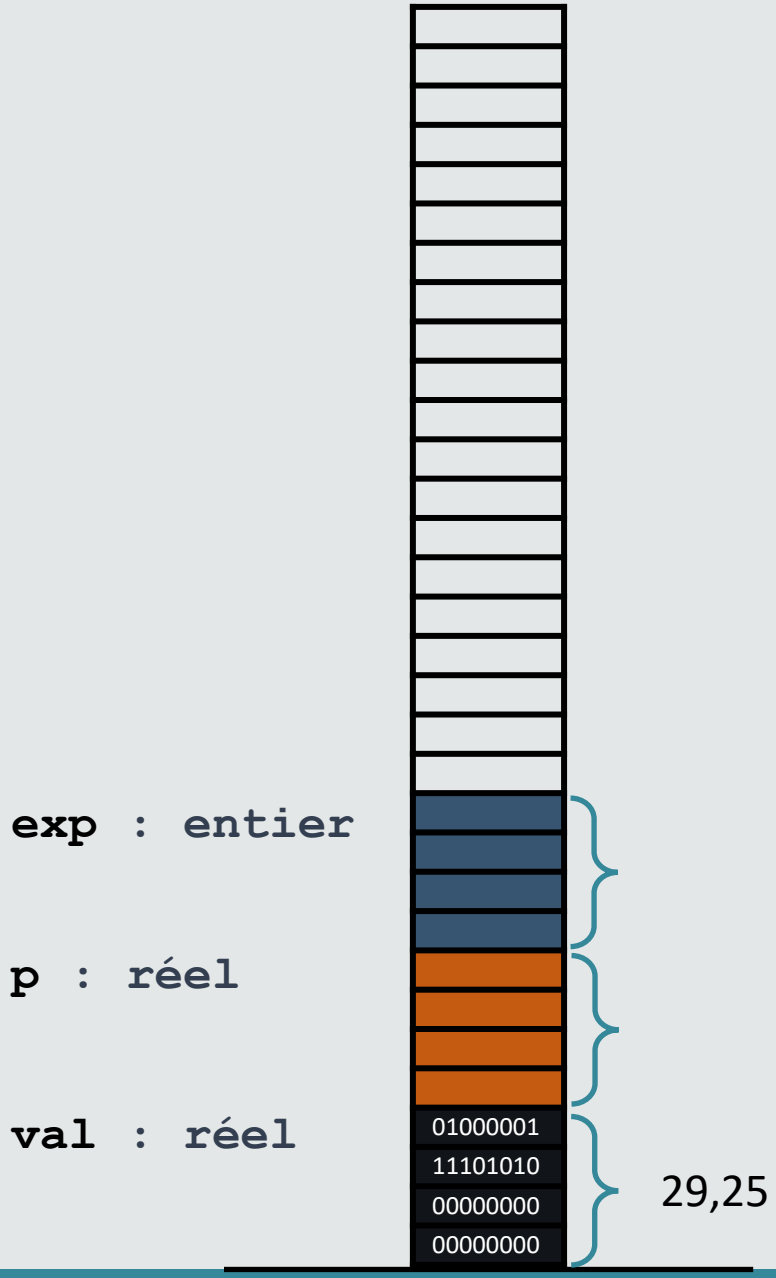
Paramètres

```
Variable  val, p      : réel
Variable  exp         : entier
```

Les procédures et fonctions

Paramètres

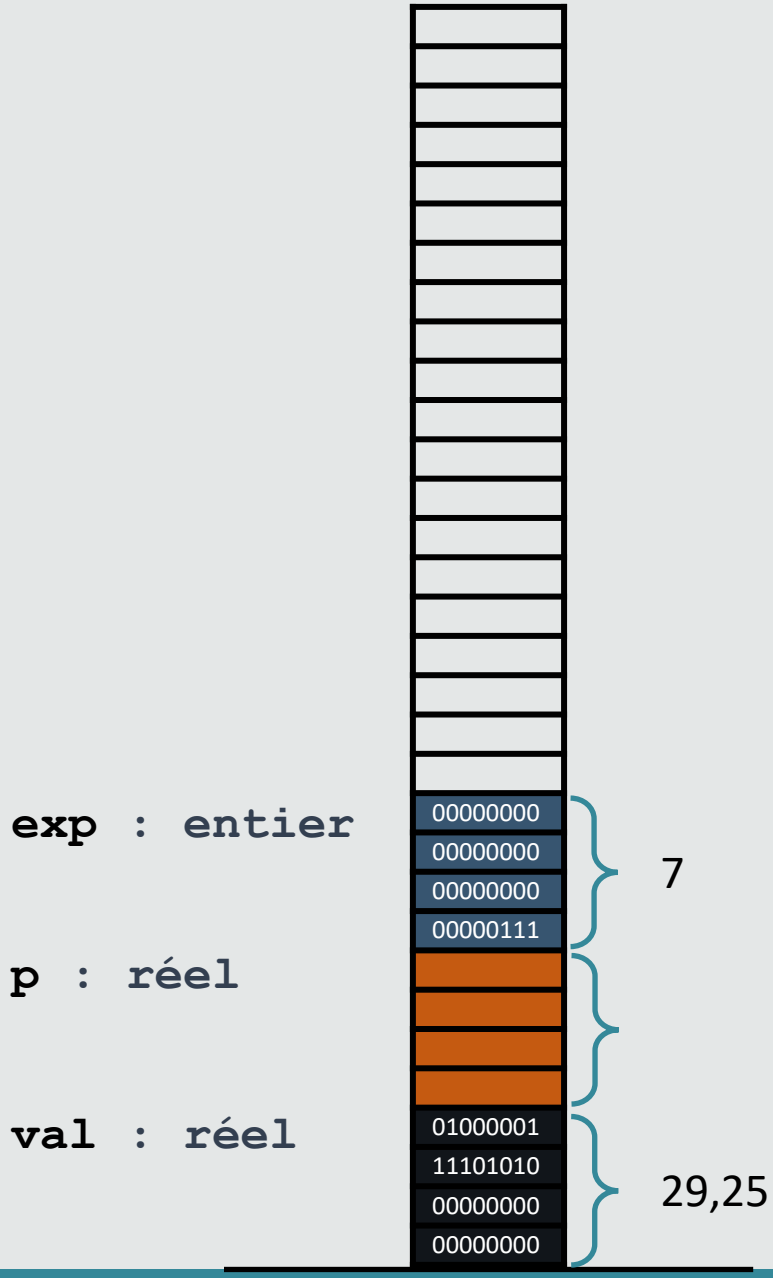
```
val <- saisir("Entrer une valeur")
29,25
```



Les procédures et fonctions

Paramètres

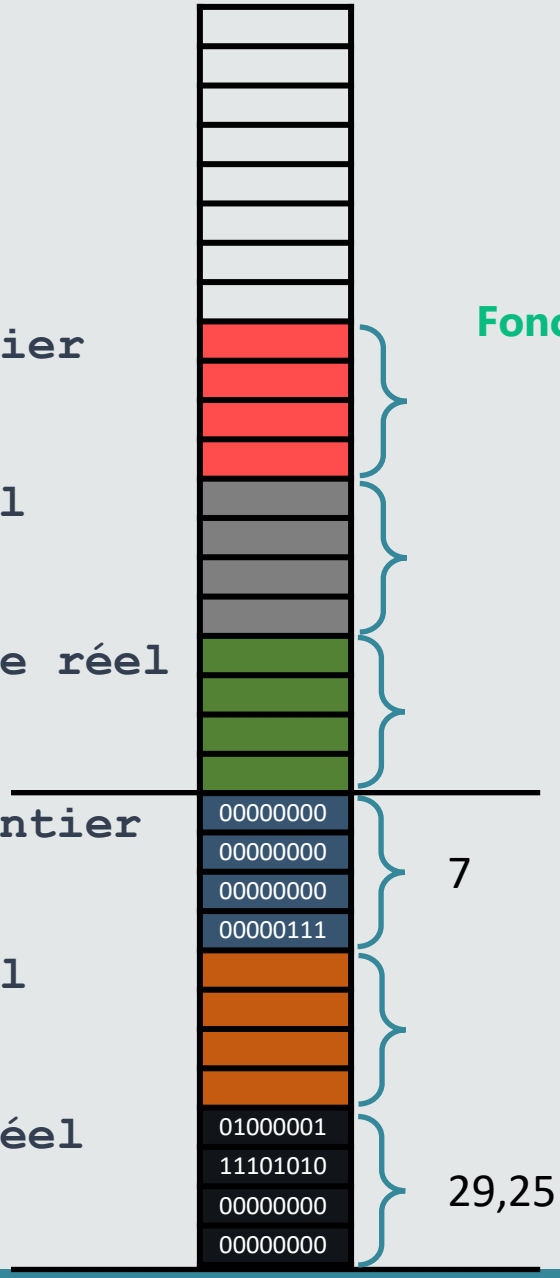
```
exp <- saisir("Entrer l'exposant")
7
```



Les procédures et fonctions

Paramètres

p <- puissance(val, exp)



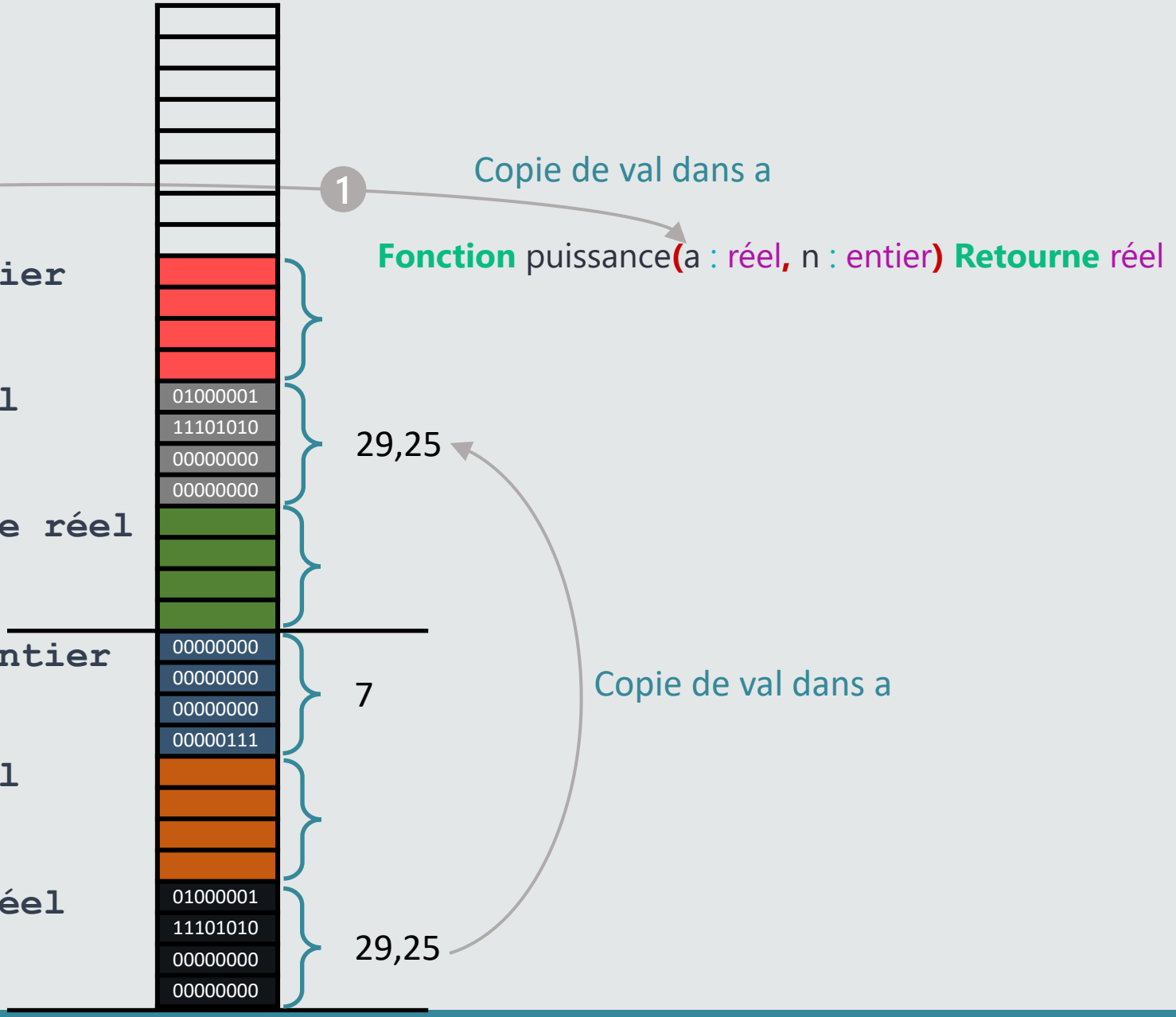
Fonction puissance(a : réel, n : entier) **Retourne** réel

Réservation de l'espace en mémoire pour le retour de la fonction et pour les paramètres

Les procédures et fonctions

Paramètres

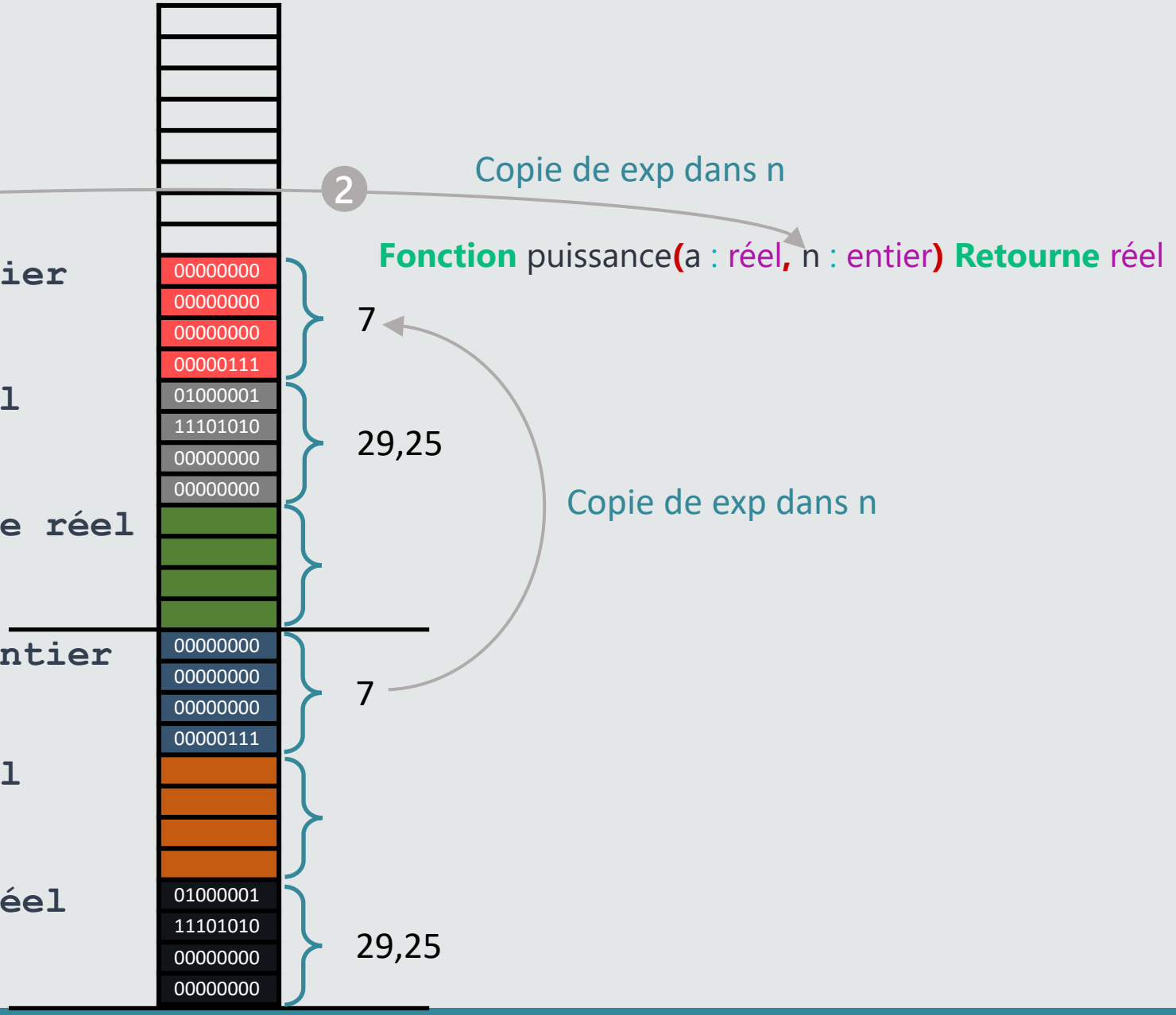
p <- puissance(val, exp)



Les procédures et fonctions

Paramètres

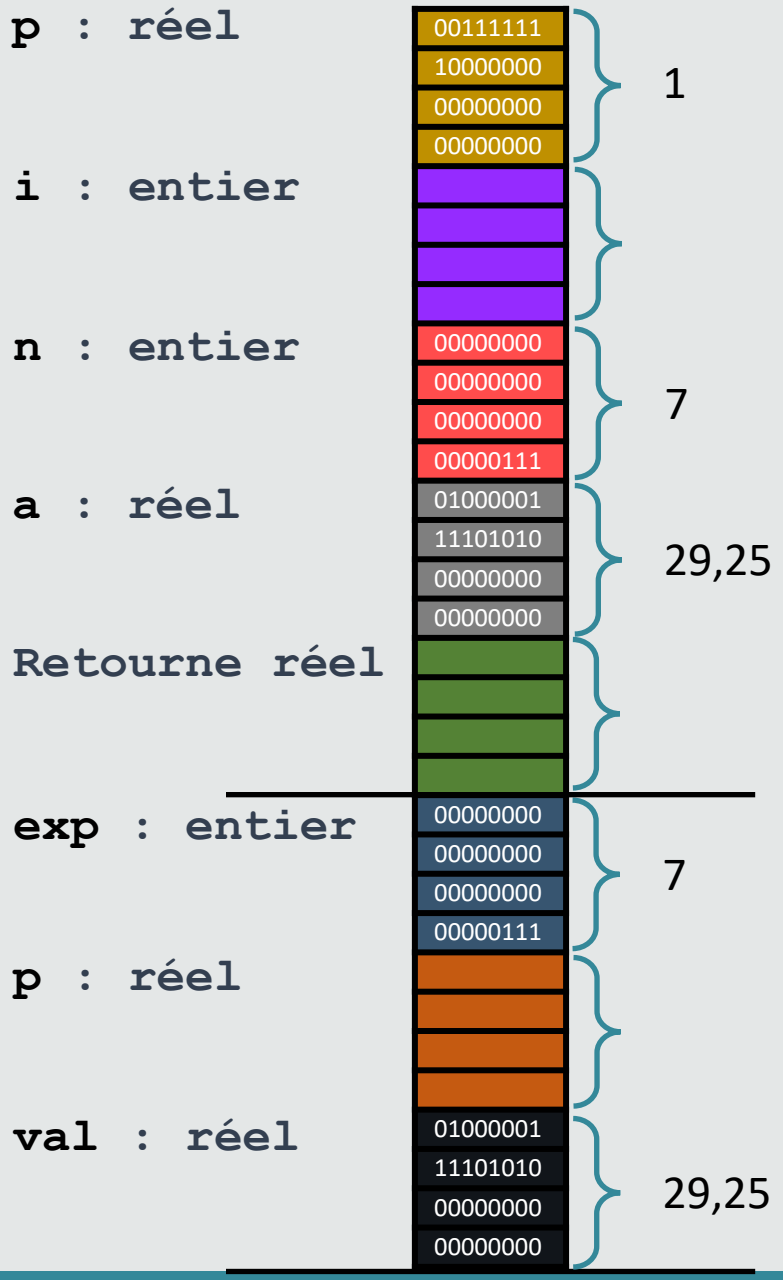
p <- puissance(val, exp)



Les procédures et fonctions

Paramètres

p <- puissance(val, exp)

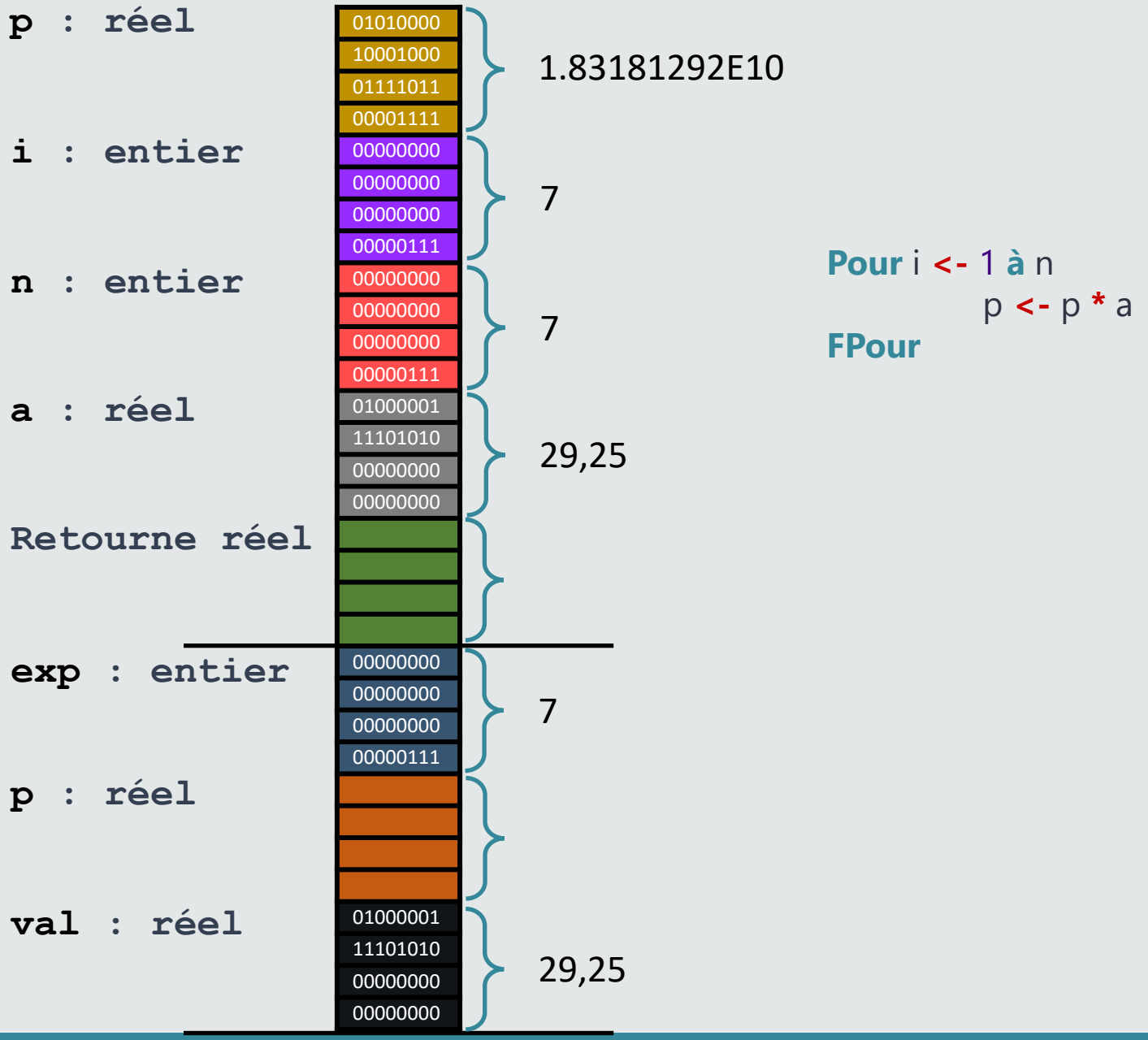


Variable i : entier
Variable p : réel <- 1

Les procédures et fonctions

Paramètres

p <- puissance(val, exp)

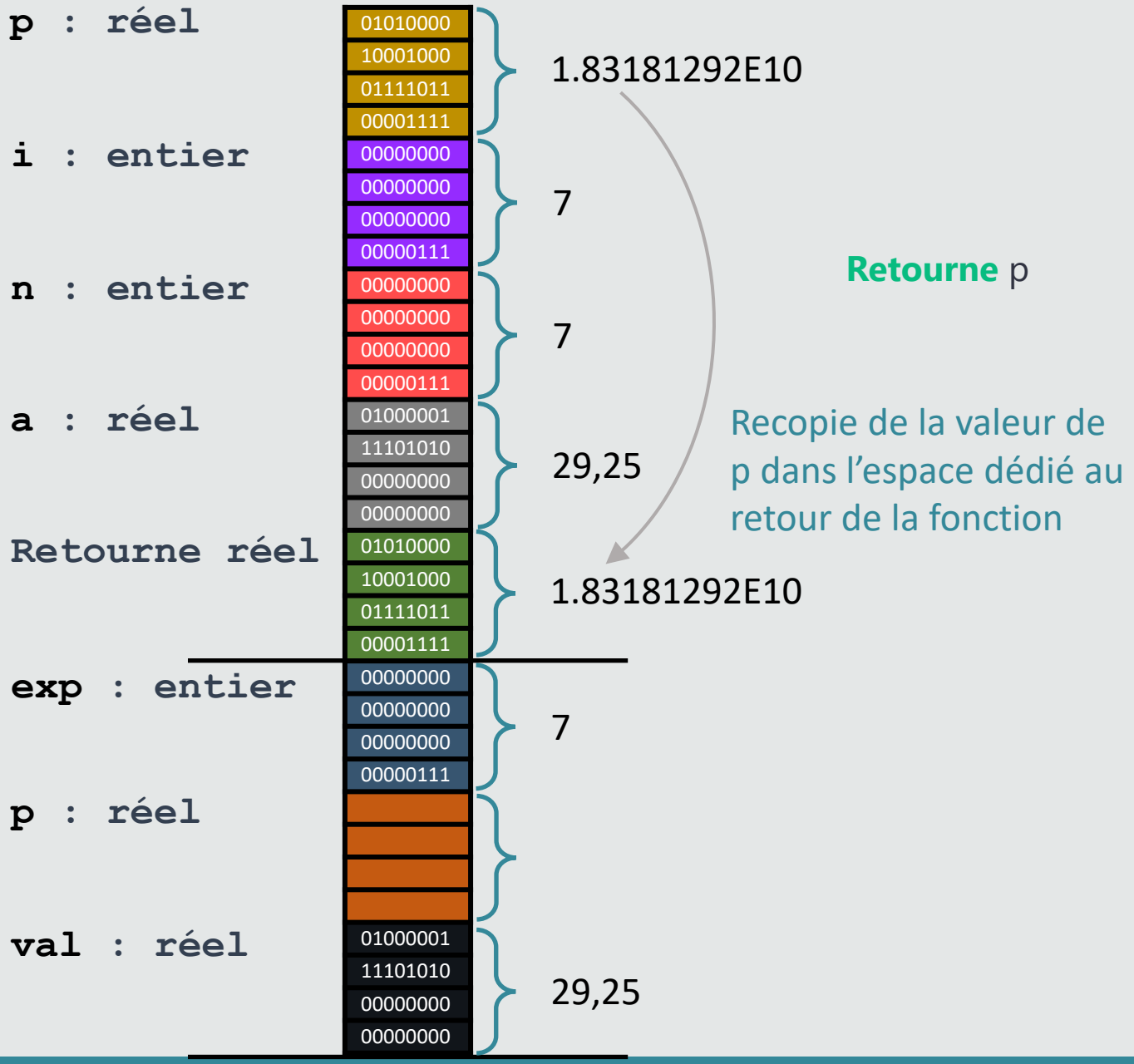



```
Pour i <- 1 à n
        p <- p * a
FPour
```

Les procédures et fonctions

Paramètres

p <- puissance(val, exp)

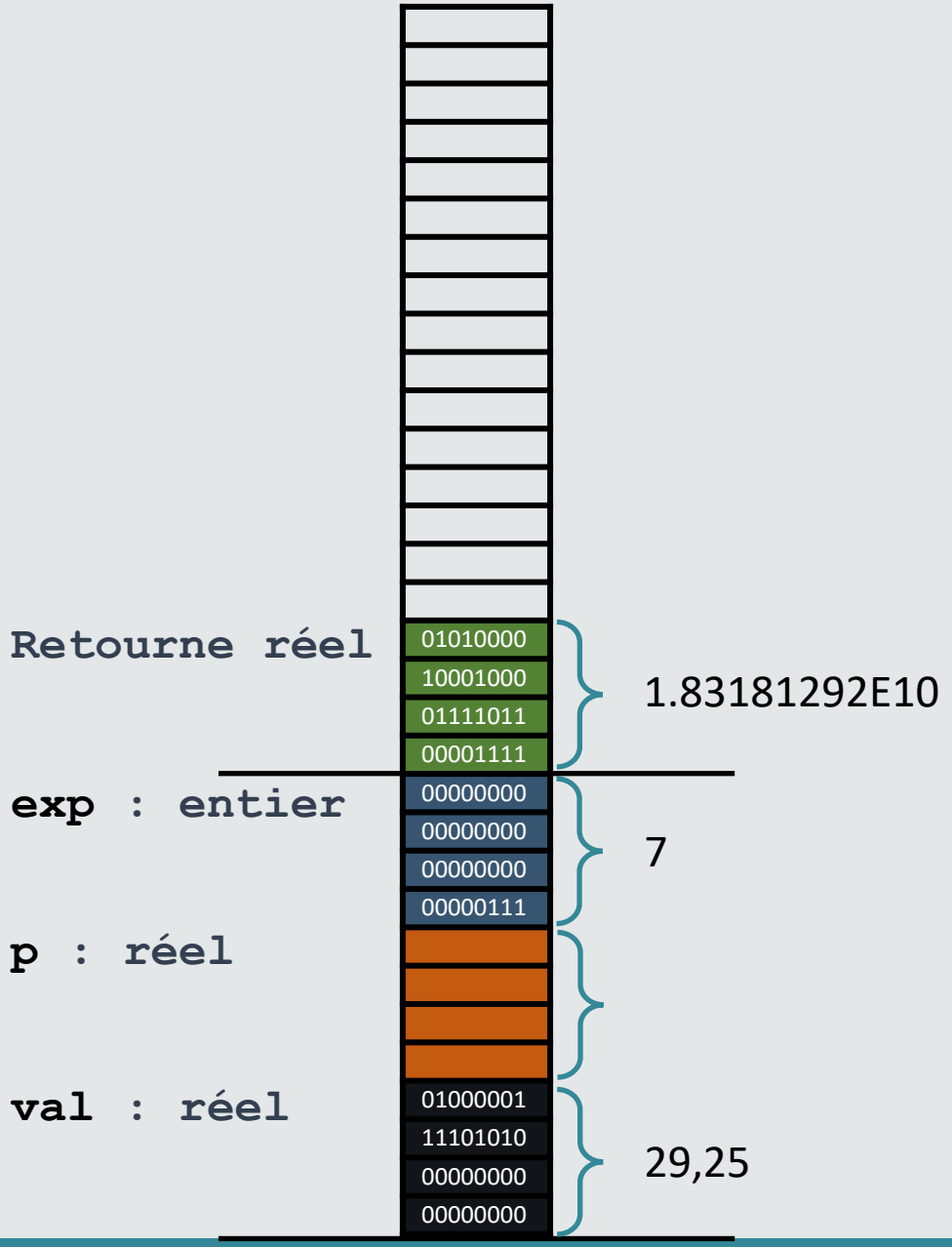


Les procédures et fonctions

Paramètres

p <- puissance(val, exp)

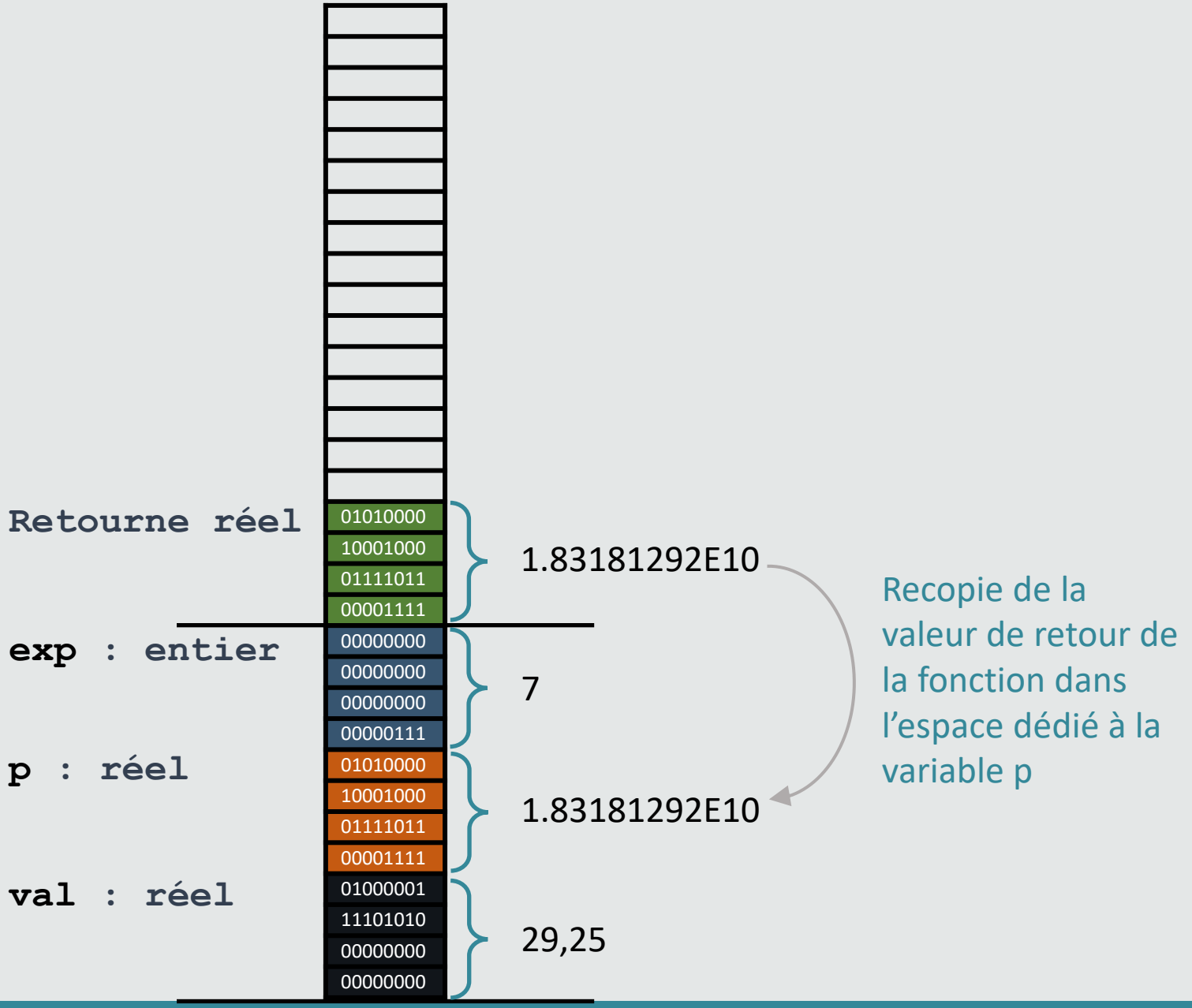
Fin # de la fonction puissance



Les procédures et fonctions

Paramètres

p <- puissance(val, exp)

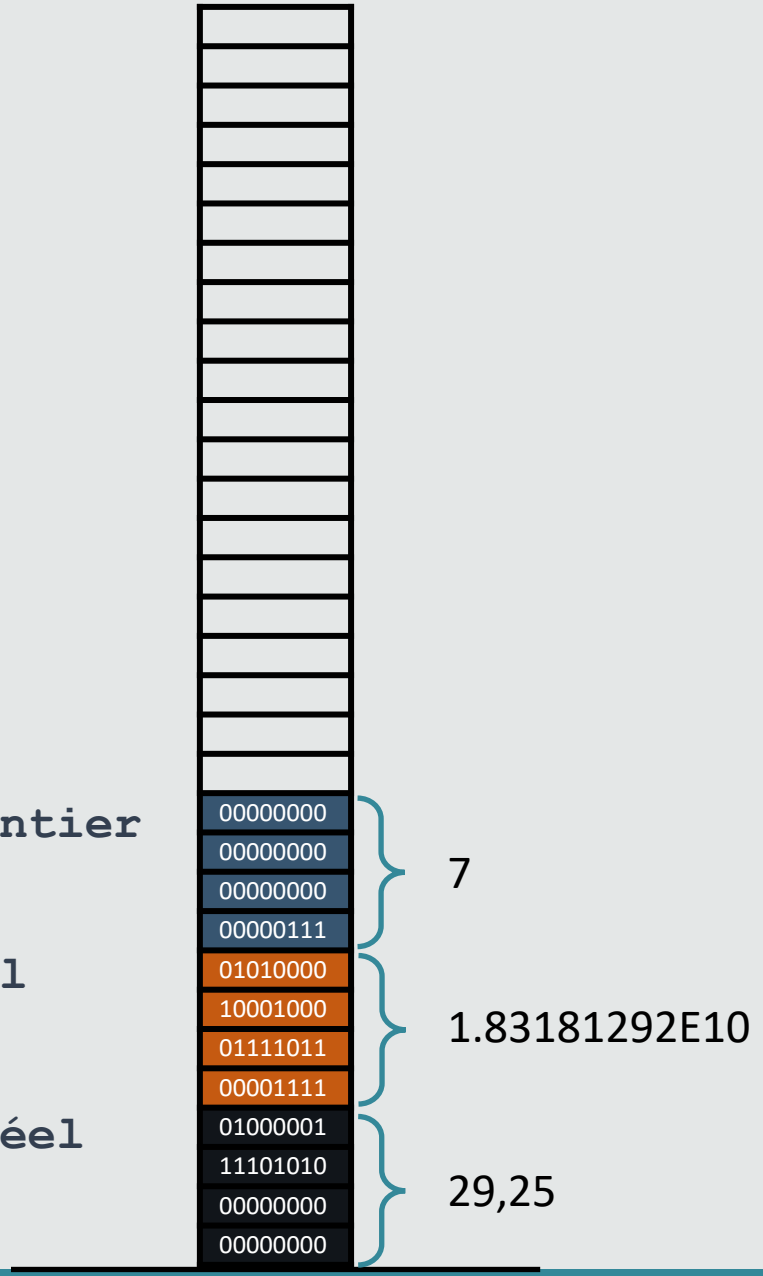


Les procédures et fonctions

Paramètres

```
    écrire(val & " puissance " & exp & " = " & p)
Fin
```

29,25 puissance 7 = 1,83181292E10



TD : C'est le plus grand

- Écrire une fonction prenant deux valeurs réelles en paramètres et qui retourne la plus grande des deux
- Écrire une autre fonction prenant également deux valeurs réelles en paramètres mais qui retourne
  - 0 si les deux valeurs sont égales
  - 1 si c'est la première valeur qui est la plus grande
  - -1 sinon
- Écrire un algorithme principal faisant appel à ces deux fonctions

Les procédures et fonctions

TD : C'est le plus grand

```
Fonction max(a : réel, b : réel) Retourne réel
# Retourne la plus grande des deux valeurs passées en paramètre
Variable plusGrand : réel
Début
    Si a > b Alors
        plusGrand <- a
    Sinon
        plusGrand <- b
    FSi
    Retourne plusGrand
Fin
```

Les procédures et fonctions

TD : C'est le plus grand

```
Fonction compare(a : réel, b : réel) Retourne entier
# Retourne une valeur pour indiquer l'ordre des deux valeurs passées en paramètre.
# 0 indique que les deux valeurs sont égales.
# 1 indique que la première valeur est la plus grande.
# -1 indique que la seconde valeur est la plus grande.
Variable ret : entier
Début
    Si a = b Alors
        ret <- 0
    Sinon
        Si a > b Alors
            ret <- 1
        Sinon
            ret <- -1
        FSi
    FSi
    Retourne ret
Fin
```

Les procédures et fonctions

TD : C'est le plus grand

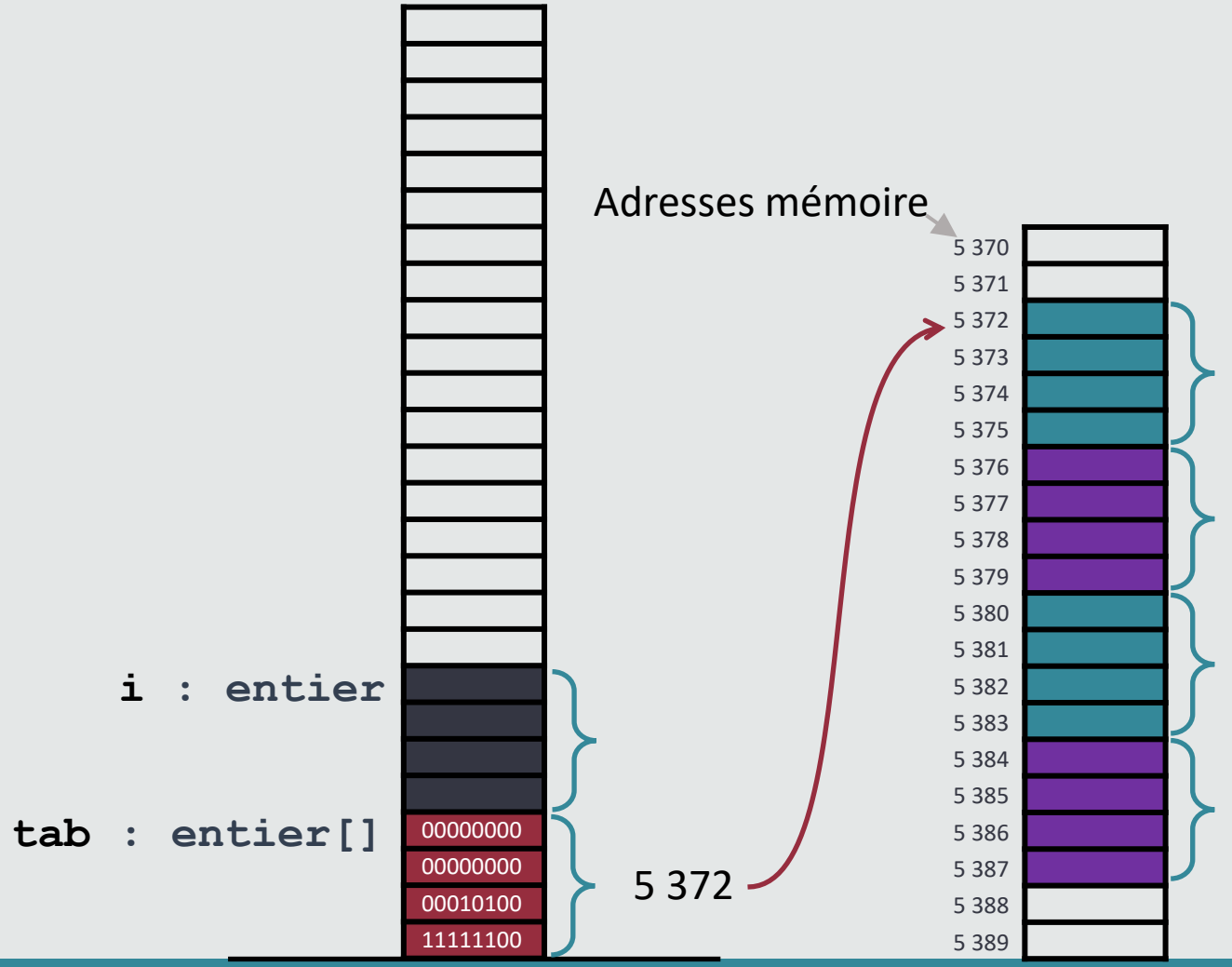
```
Algo testFonctions
# Teste les fonctions compare et max
Variable val1, val2, maximum : réel
Variable rep : entier
Début
    val1 <- saisir("Entrer une première valeur")
    val2 <- saisir("Entrer une seconde valeur")
    maximum <- max(val1, val2)
    écrire("Le maximum de " & val1 & " et " & val2 & " est " & maximum)
    rep <- compare(val1, val2)
    Si rep = 0 Alors
        écrire("Les deux valeurs sont égales")
    Sinon
        Si rep > 0 Alors
            écrire("La première valeur est la plus grande")
        Sinon
            écrire("La seconde valeur est la plus grande")
        FSi
    FSi
Fin
```

Le passage d'un tableau en paramètre

```
Procédure multi(tableau : entier[], taille : entier, coef : entier)
# Multiplie tous les éléments d'un tableau par un coefficient
Variable i : entier
Début
    Pour i <- 0 à taille – 1
        tableau[i] <- tableau[i] × coef
    FPour
Fin # Procédure multi
Algo testMulti # Test la procédure « multi »
Constante TAILLE : entier <- 4
Variable tab : entier[TAILLE]
Variable i : entier
Début
    Pour i <- 0 à TAILLE – 1
        tab[i] <- saisir("Entrer valeur n°" & i + 1)
    FPour
    multi(tab, TAILLE, 2)
    Pour i <- 0 à TAILLE – 1
        écrire("Valeur n°" & i + 1 & " = " & tab[i])
    FPour
Fin
```

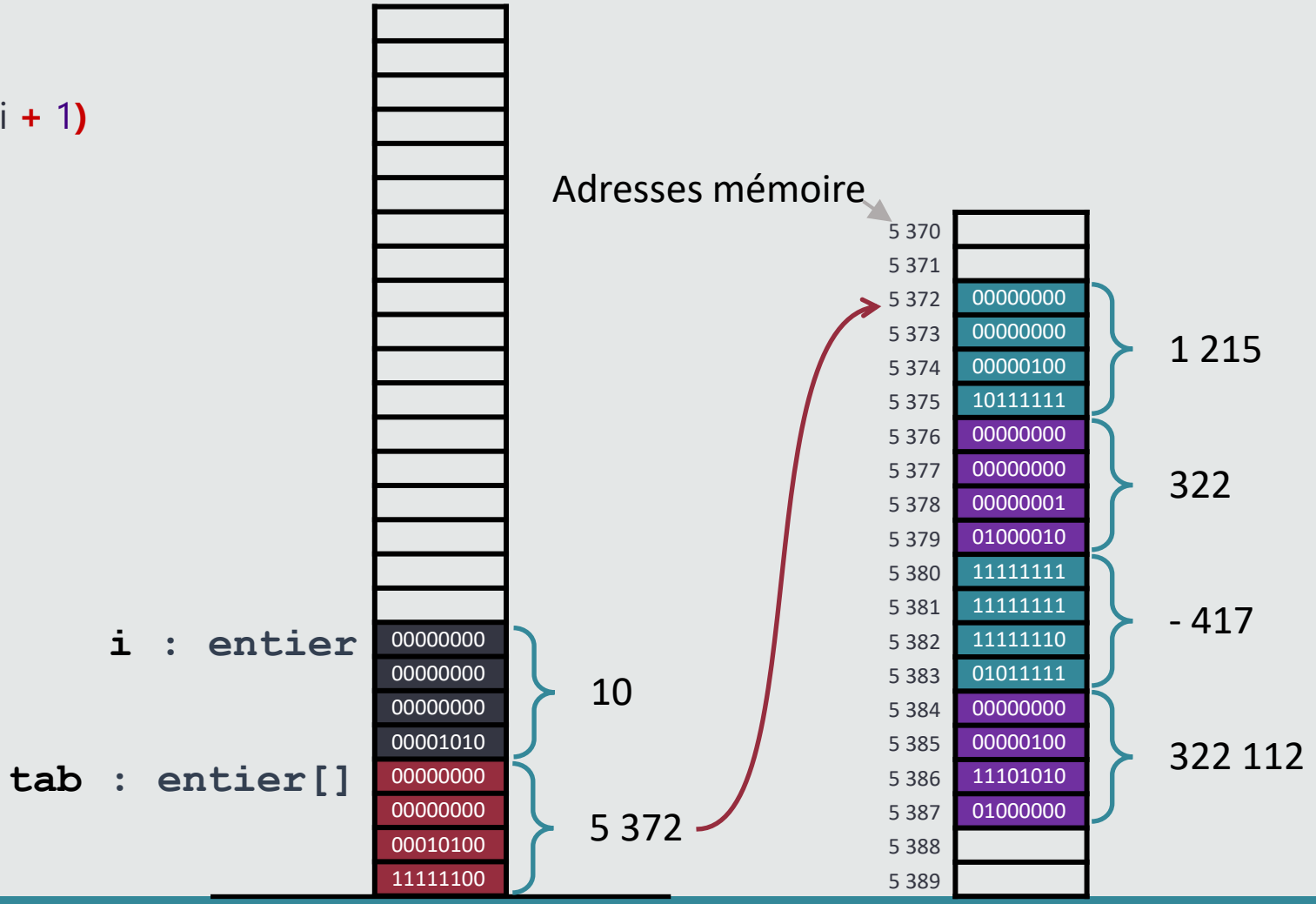
Le passage d'un tableau en paramètre

```
Constante TAILLE : entier <- 4
Variable tab : entier[TAILLE]
Variable i : entier
```



Le passage d'un tableau en paramètre

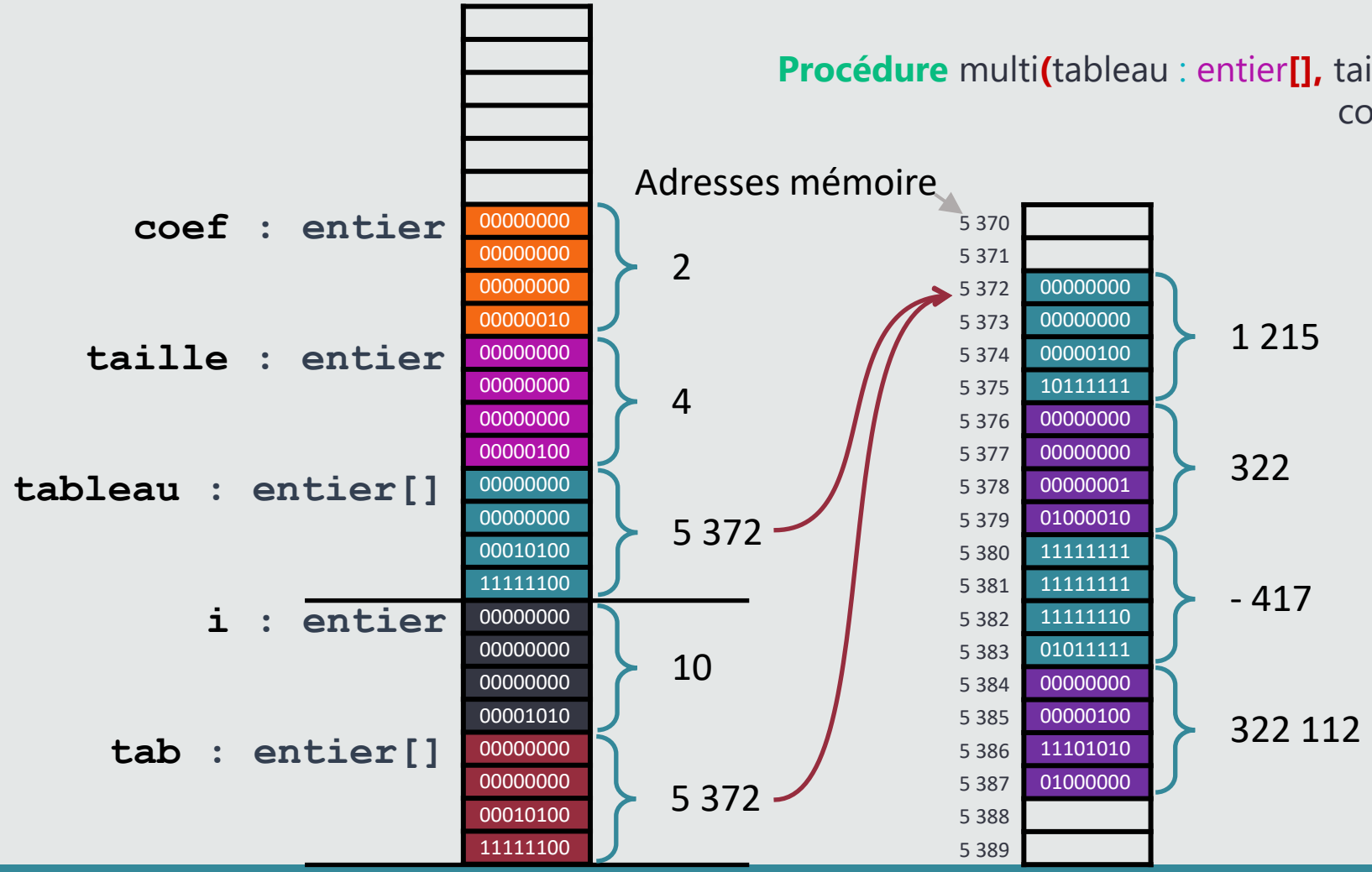
```
Pour i <- 0 à TAILLE – 1
    tab[i] <- saisir("Entrer valeur n°" & i + 1)
FPour
```



Le passage d'un tableau en paramètre

multi(tab, TAILLE, 2)

Procédure multi(tableau : entier[], taille : entier, coef : entier)



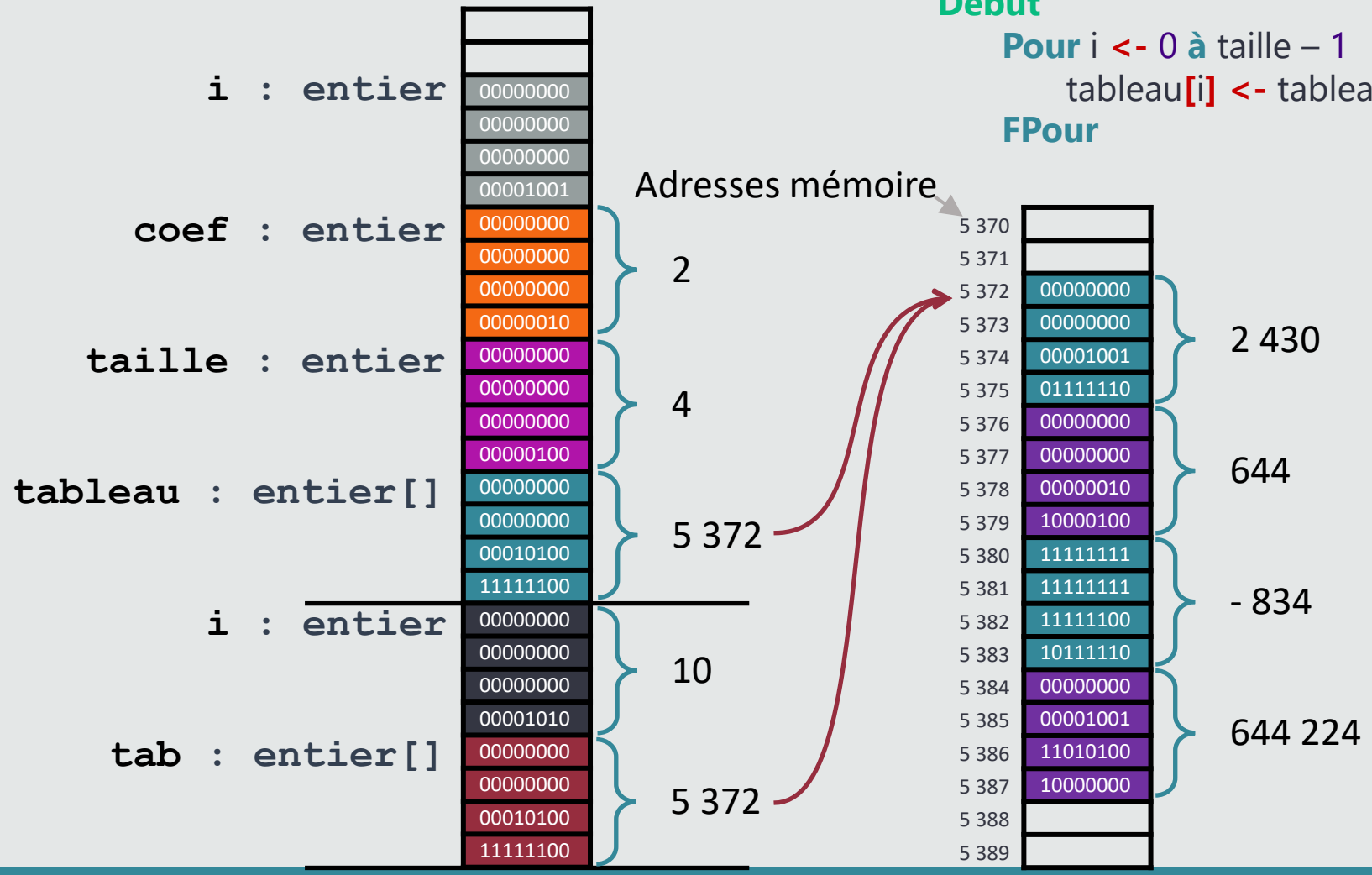
Recopie des paramètres

Attention : seule l'adresse mémoire du tableau est recopiée

Le passage d'un tableau en paramètre

multi(tab, TAILLE, 2)

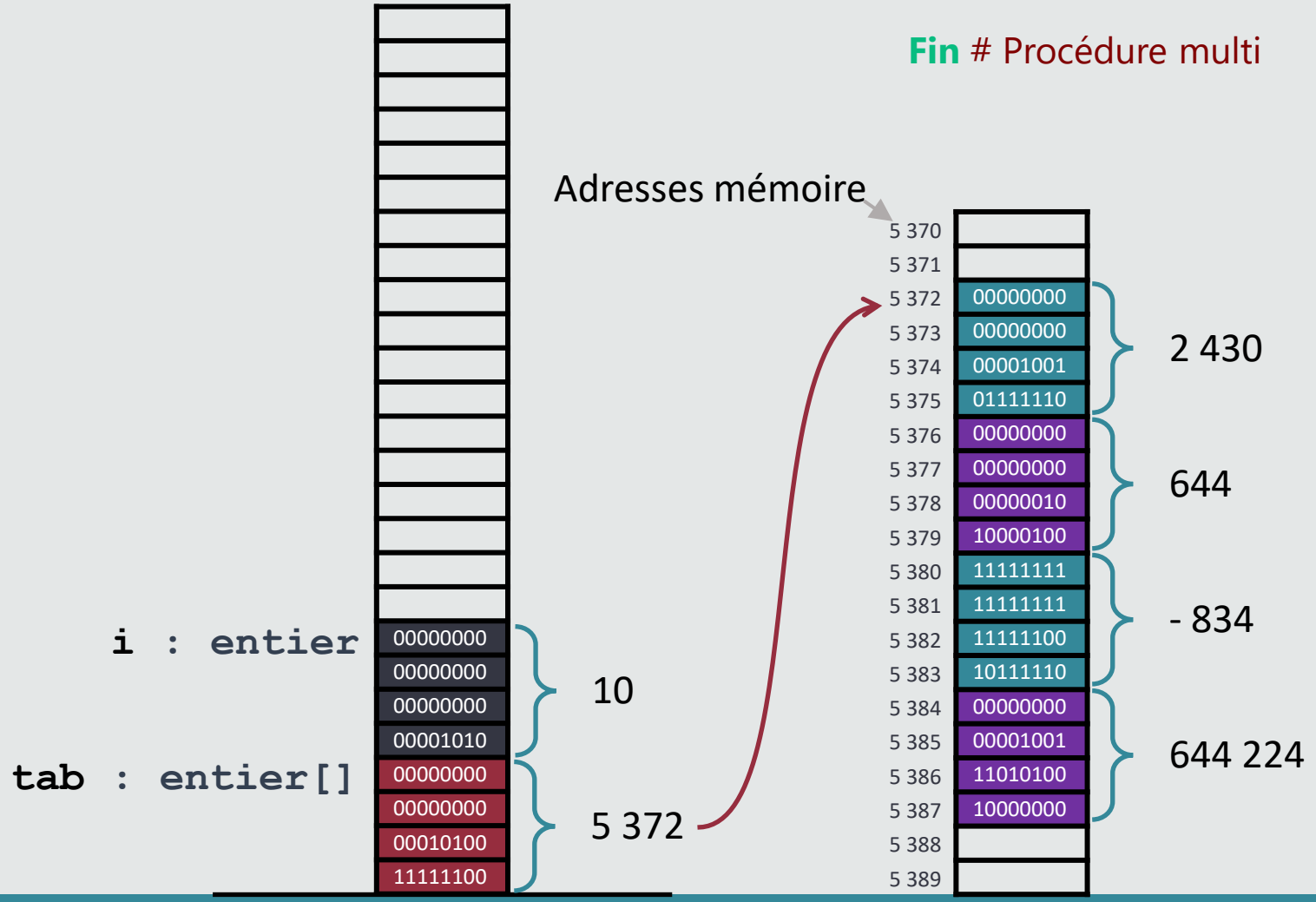

```
Variable i : entier
Début
    Pour i <- 0 à taille – 1
        tableau[i] <- tableau[i] × coef
    FPour
```



Le passage d'un tableau en paramètre

multi(tab, TAILLE, 2)

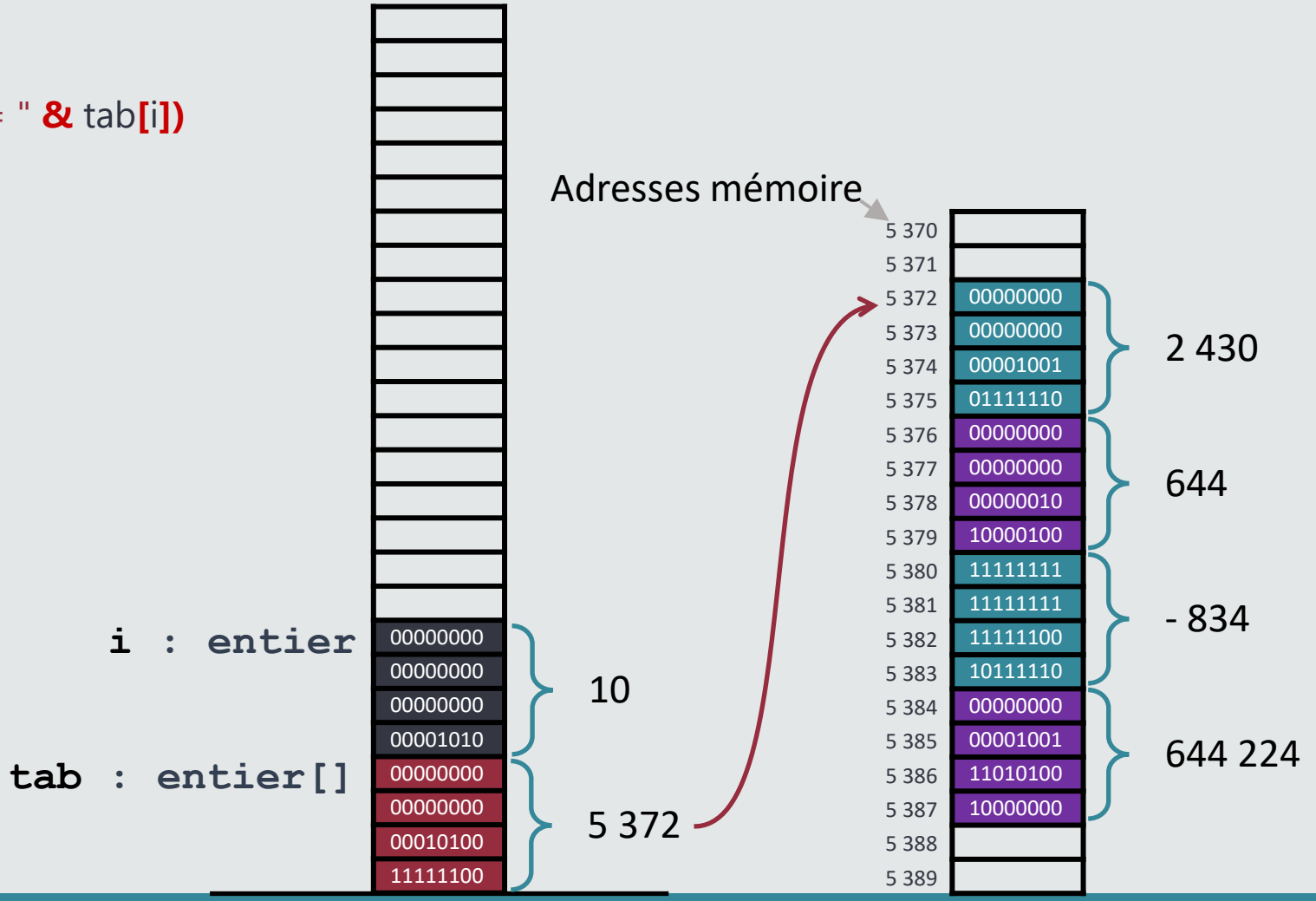
Fin # Procédure multi



Le passage d'un tableau en paramètre

```
Pour i <- 0 à TAILLE – 1
    écrire("Valeur n°" & i + 1 & " = " & tab[i])
FPour
Fin
```

Valeur n°1 = 2430
Valeur n°2 = 644
Valeur n°3 = -834
Valeur n°4 = 644224



Les constantes globales

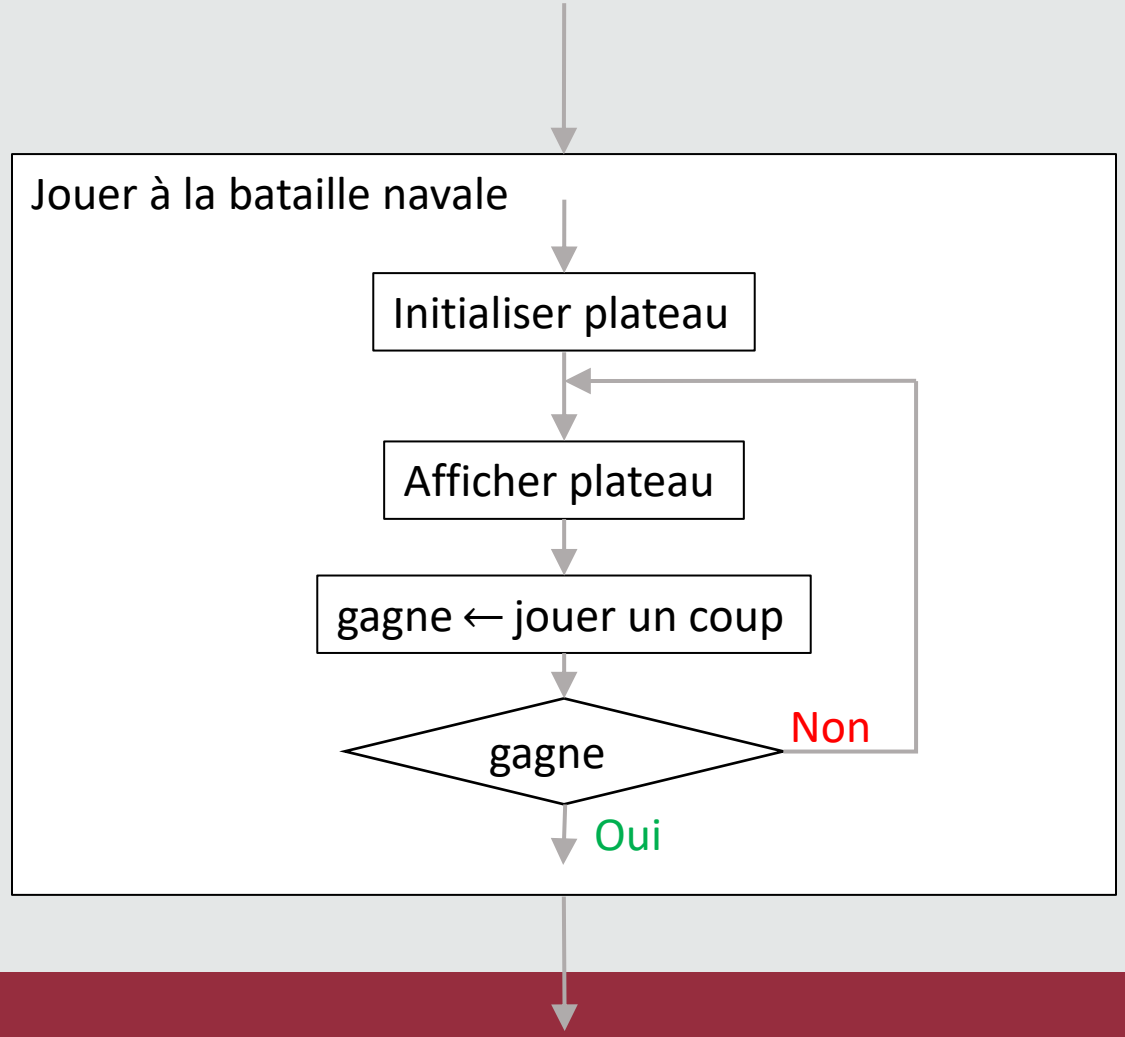
- Une variable peut être définie au niveau global
  - accessible dans l'algorithme principal et dans toutes les fonctions

```
Constante GLOBALE : entier <- 42
Algo testConstantes
Constante LOCAL : entier <- -77
Début
        écrire("il est possible d'accéder à la constante GLOBALE : " & GLOBALE &
                " et à la variable LOCAL " & LOCAL)
        proc()
Fin

Procédure proc()
Début
        écrire("il est possible d'accéder à la constante GLOBALE : " & GLOBALE &
                " mais pas à la variable LOCAL")
Fin
```

TD : Micro bataille navale (version 2)

- Structuration



Les procédures et fonctions

TD : Micro bataille navale (version 2)

```
Constante PLOUF      : caractère <- '~'
Constante BATEAU     : caractère <- 'b'
Constante EAU        : caractère <- 'e'

Algo microBatailleNavaleV2
# Jeu de la Bataille Navale avec un seul bateau d'une case
Constante HAUTEUR    : entier <- 4
Constante LARGEUR    : entier <- 4
Variable  plateau    : caractère[HAUTEUR][LARGEUR]
Variable  gagne      : booléen
Début
    # Initialisation du plateau de jeu
    initialiserPlateau(plateau, HAUTEUR, LARGEUR)
    Répéter
        # Affichage du plateau de jeu
        afficherPlateau(plateau, HAUTEUR, LARGEUR)
        # Le joueur tire un coup
        gagne <- tirer(plateau, HAUTEUR, LARGEUR)
    TantQue Non gagne FRépéter
Fin
```

Les procédures et fonctions

TD : Micro bataille navale (version 2)

```
Procédure initialiserPlateau(plateau : caractère[][], dim1 : entier, dim2 : entier)
# Initialise le plateau de jeu avec de l'eau dans toutes les cases
# sauf une qui contient le bateau
Variable i, j : entier
Début
    # Toutes les cases du plateau sont initialisées à EAU
    Pour j <- 0 à dim1-1
        Pour i <- 0 à dim2-1
            plateau[j][i] <- EAU
        FPour
    FPour
    # Dépôt d'un bateau d'une case sur une case choisie aléatoirement
    plateau[aléa(0,dim1-1)][aléa(0,dim2-1)] <- BATEAU
Fin
```

Les procédures et fonctions

TD : Micro bataille navale (version 2)

```
Procédure afficherPlateau(plateau : caractère[][], dim1 : entier, dim2 : entier)
# Affichage du plateau de jeu
Variable i, j : entier
Début
    Pour j <- 0 à dim1-1
        Pour i <- 0 à dim2-1
            Si plateau[j][i] = BATEAU Ou plateau[j][i] = EAU Alors
                écrireSRC("?")
            Sinon
                écrireSRC(plateau[j][i])
            FSi
        FPour
        écrire()
    FPour
Fin
```

Les procédures et fonctions

TD : Micro bataille navale (version 2)

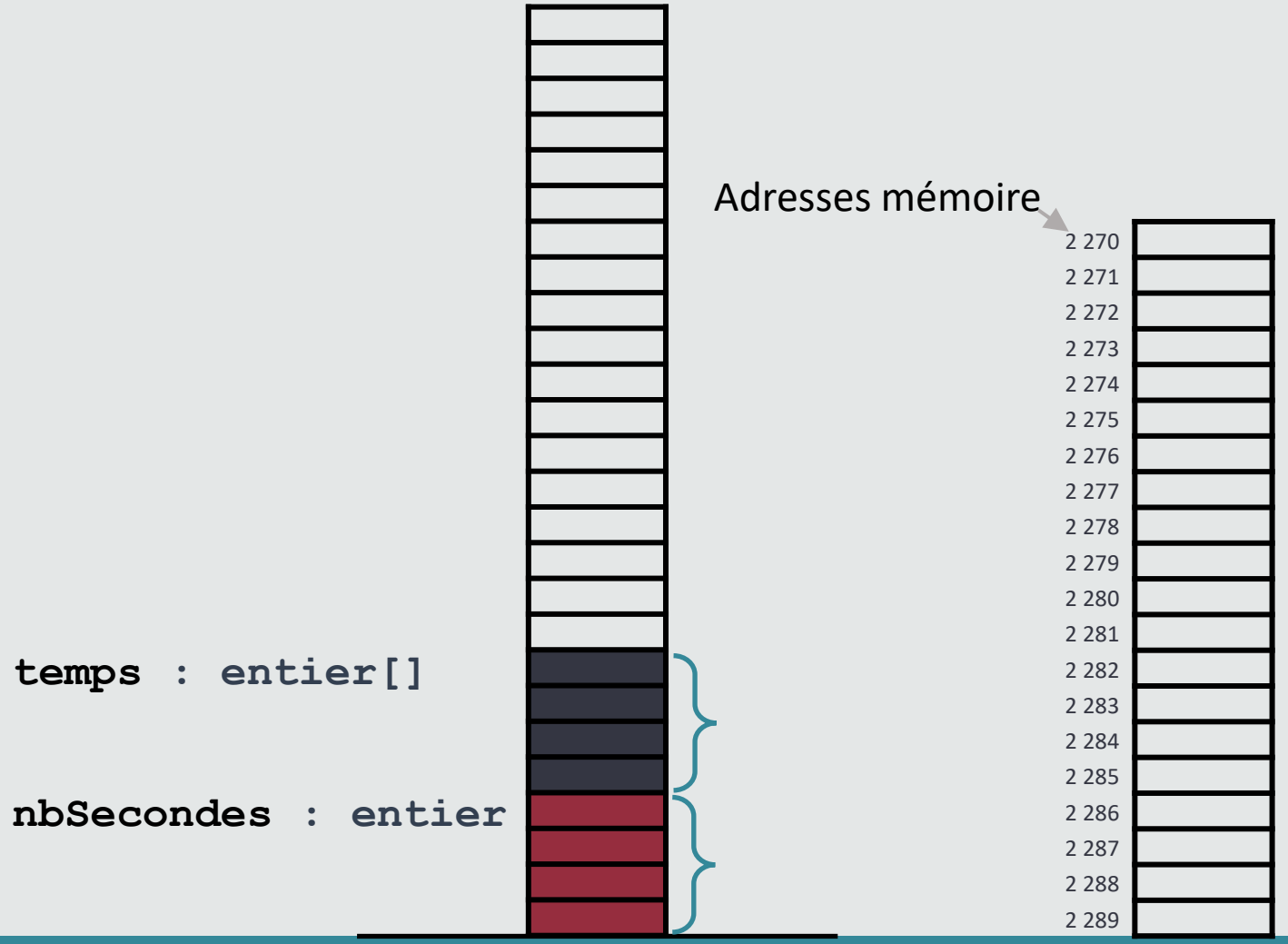
```
Fonction tirer(plateau : caractère[][], dim1 : entier, dim2 : entier) Retourne Booléen
# le joueur joue un coup
Variable x, y : entier
Début
    Répéter                                  # Saisie des coordonnées de tir
        x <- saisir("Quelle colonne (entre 1 et " & dim2 & ") : ") - 1
        y <- saisir("Quelle ligne (entre 1 et " & dim1 & ") : ") - 1
    TantQue x < 0 ou x ≥ dim2 ou y < 0 ou y ≥ dim1 FRépéter
    Si plateau[y][x] ≠ BATEAU Alors          # test du tir
        plateau[y][x] <- PLOUF
        écrire("Plouf !")
    Sinon
        écrire("Touché Coulé ! Bravo, vous avez gagné !")
    FSi
    Retourne plateau[y][x] = BATEAU
Fin
```

Retourner un tableau

```
Constante H : entier <- 0
Constante S : entier <- 2
Constante M : entier <- 1
Constante TAILLE : entier <- 3
Algo testNbSecToHMS
Variable nbSecondes : entier
Variable temps : entier[]
Début
    nbSecondes <- saisir("Entrer une durée en secondes")
    temps <- nbSecToHMS(nbSecondes)
    écrire("durée = " & temps[H] & ":" & temps[M] & ":" & temps[S])
Fin
Fonction nbSecToHMS(nbS : entier) Retourne entier[]
# convertit un temps en secondes en heures, minutes, secondes
Variable hms : entier[TAILLE]
Début
    hms[H] <- nbS div 3600
    nbS <- nbS % 3600
    hms[M] <- nbS div 60
    hms[S] <- nbS % 60
    Retourner hms
Fin # Fonction NbSecToHMS
```

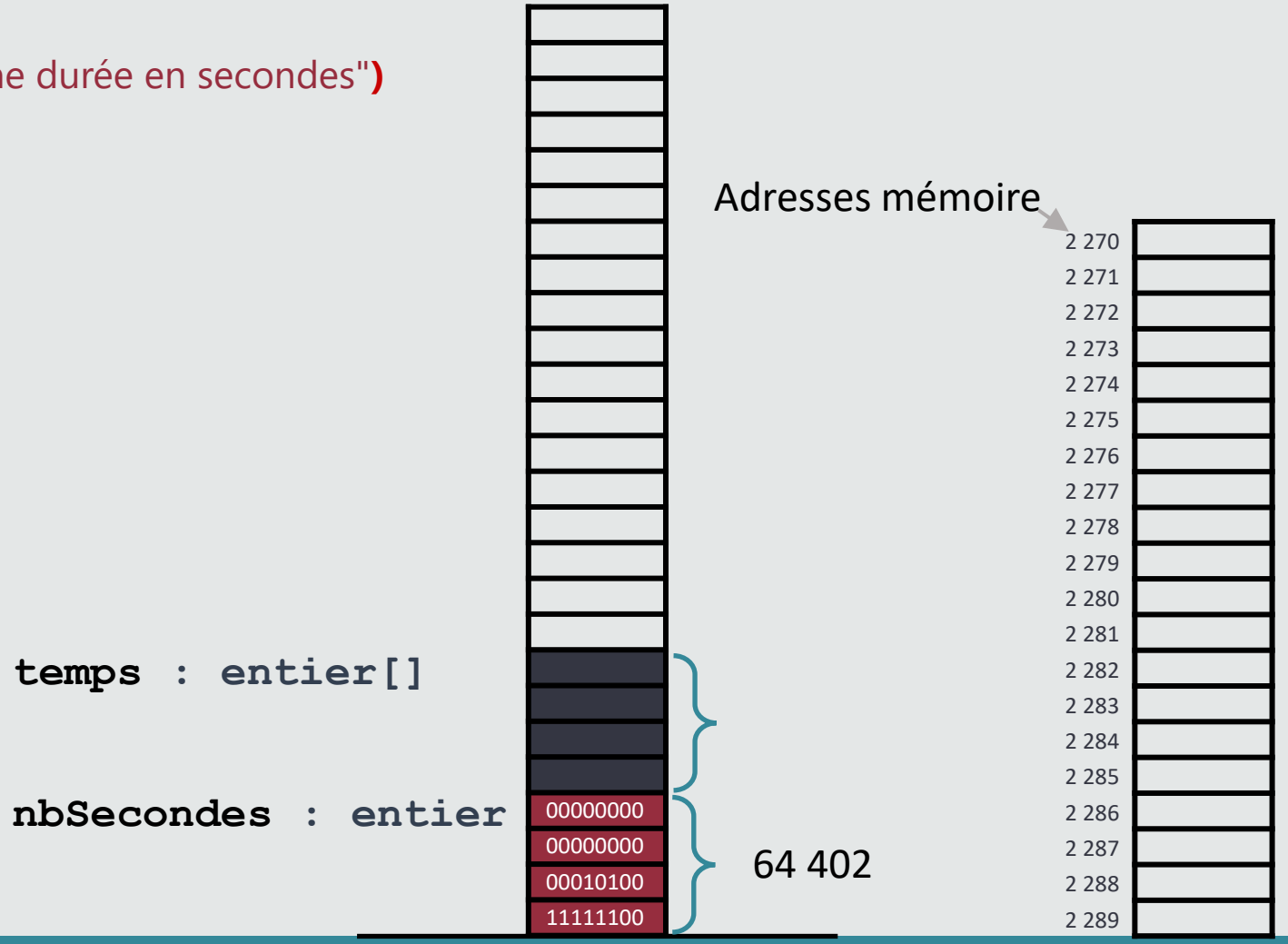
Retourner un tableau

```
Algo testNbSecToHMS
Variable nbSecondes : entier
Variable temps : entier[]
```



Retourner un tableau

```
nbSecondes <- saisir("Entrer une durée en secondes")
64402
```

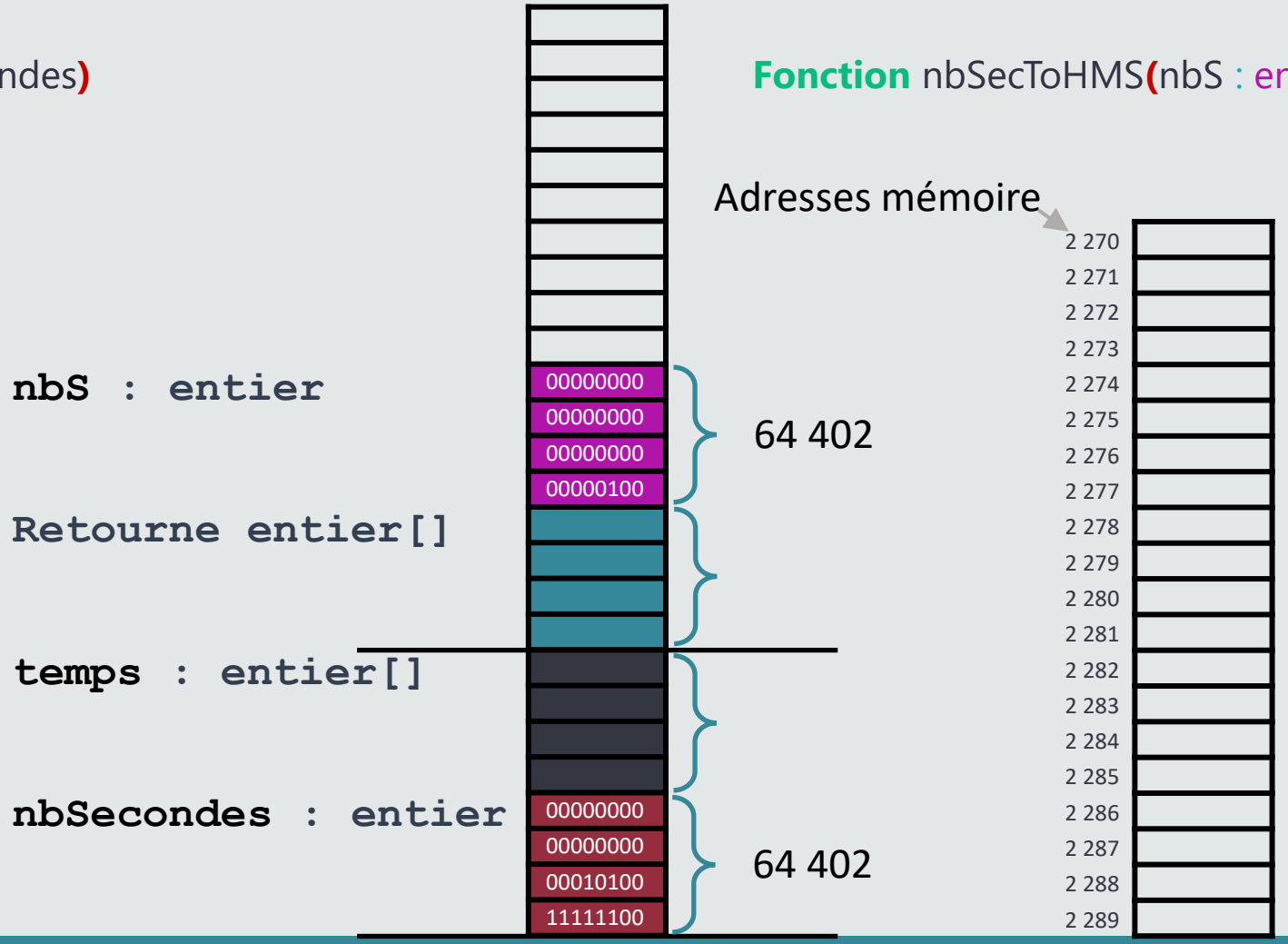


Les procédures et fonctions

Retourner un tableau

temps <- nbSecToHMS(nbSecondes)

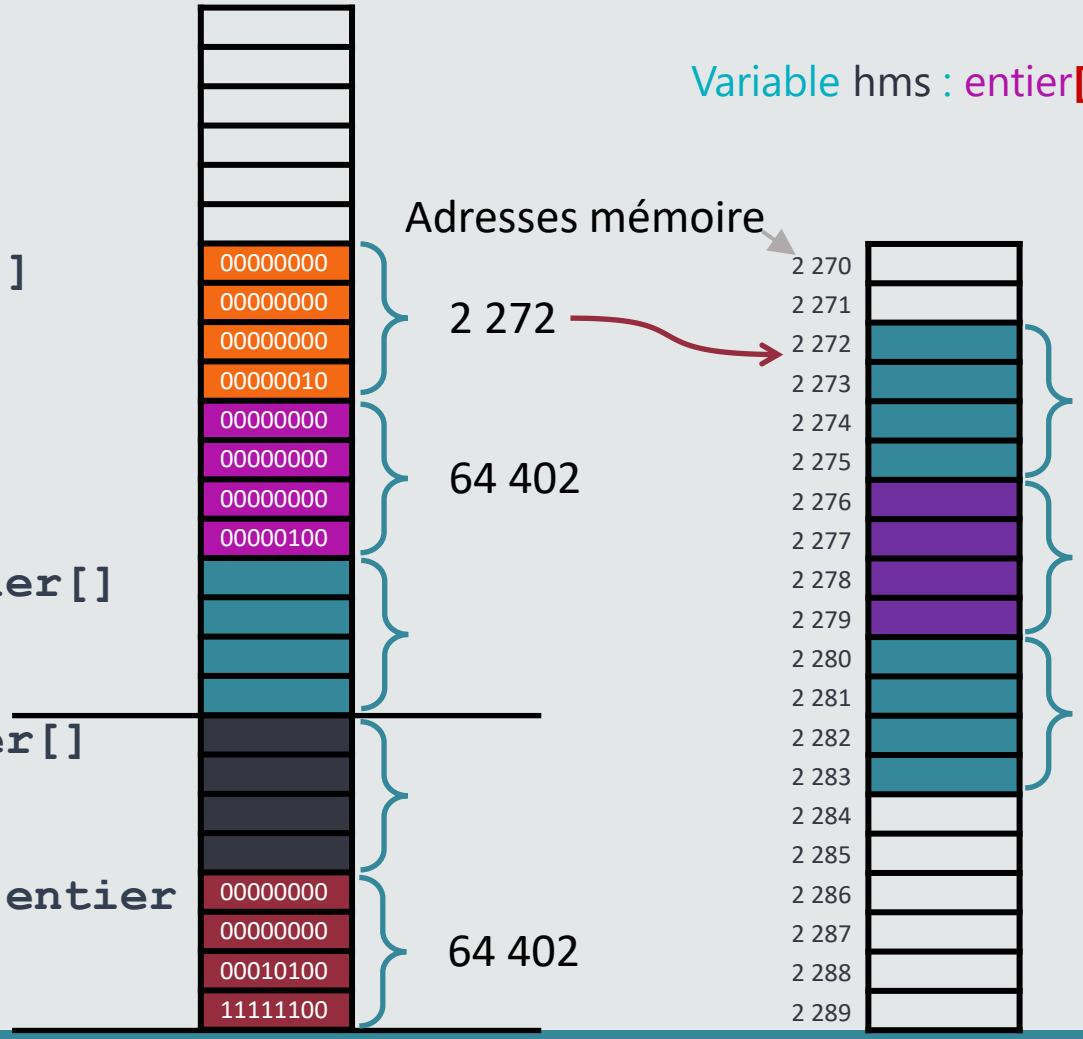
Fonction nbSecToHMS(nbS : entier) **Retourne** entier[]



Les procédures et fonctions

Retourner un tableau

temps <- nbSecToHMS(nbSecondes)

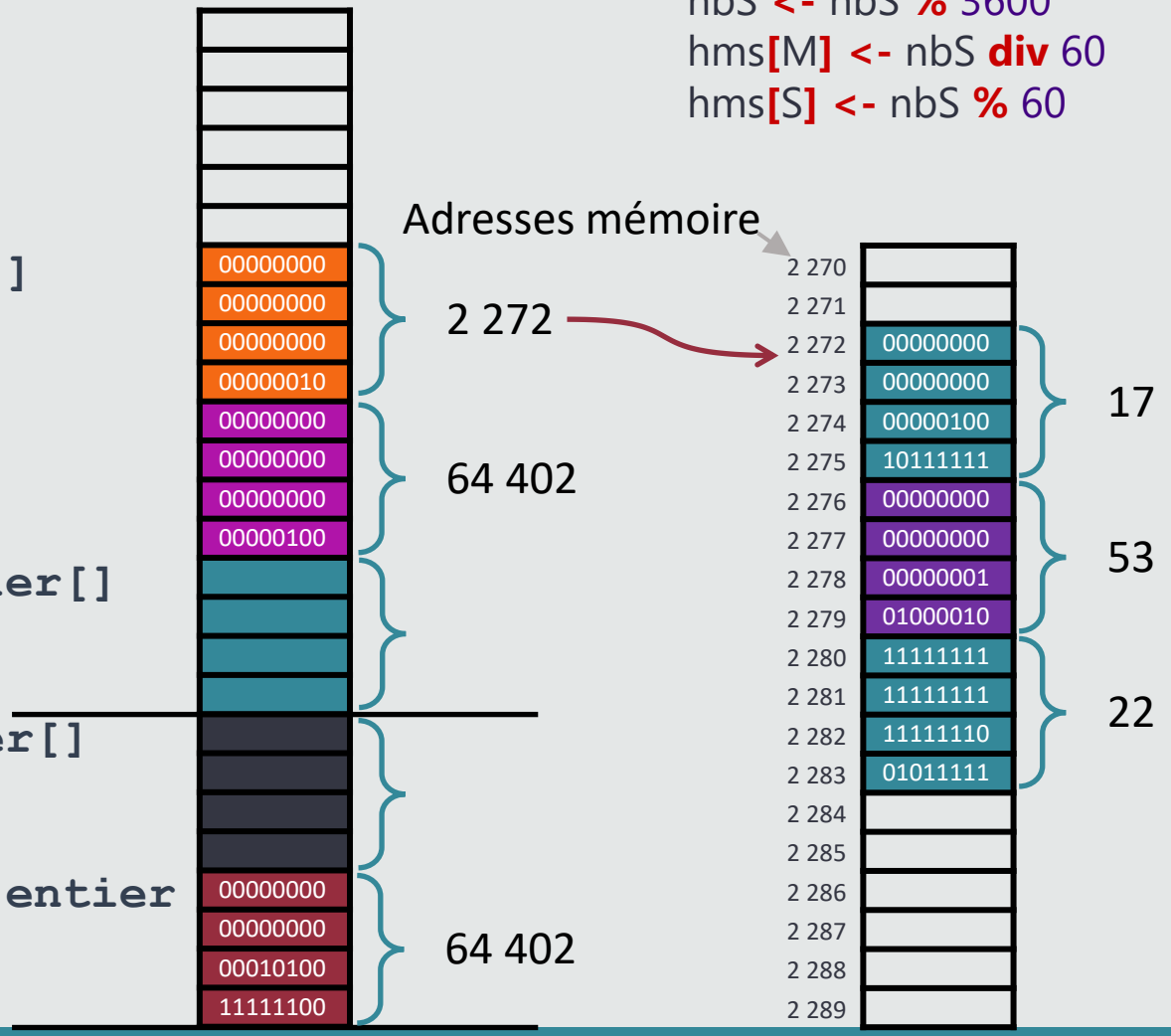


Les procédures et fonctions

Retourner un tableau

temps <- nbSecToHMS(nbSecondes)

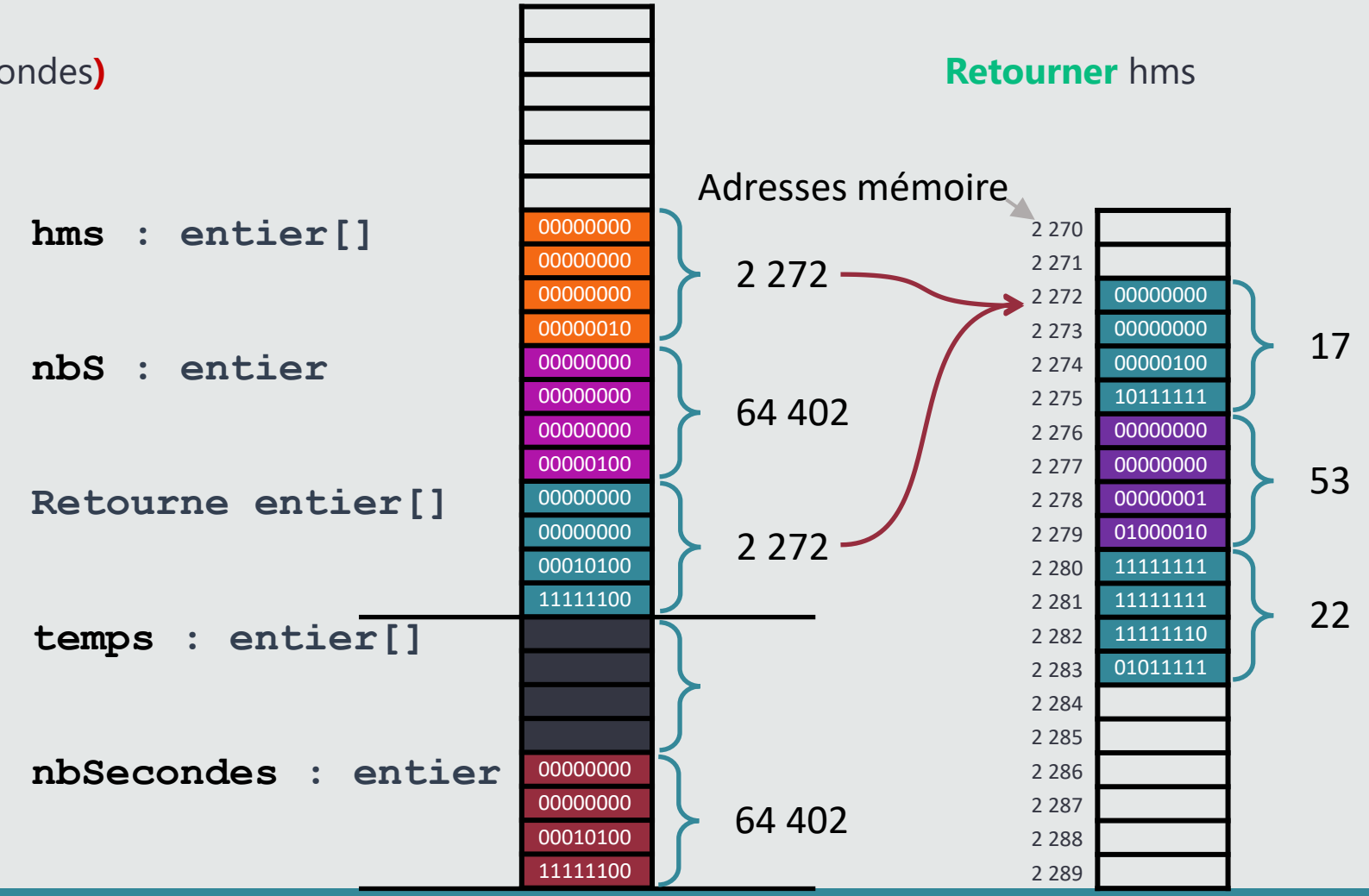
hms[H] <- nbS **div** 3600
nbS <- nbS **%** 3600
hms[M] <- nbS **div** 60
hms[S] <- nbS **%** 60



Les procédures et fonctions

Retourner un tableau

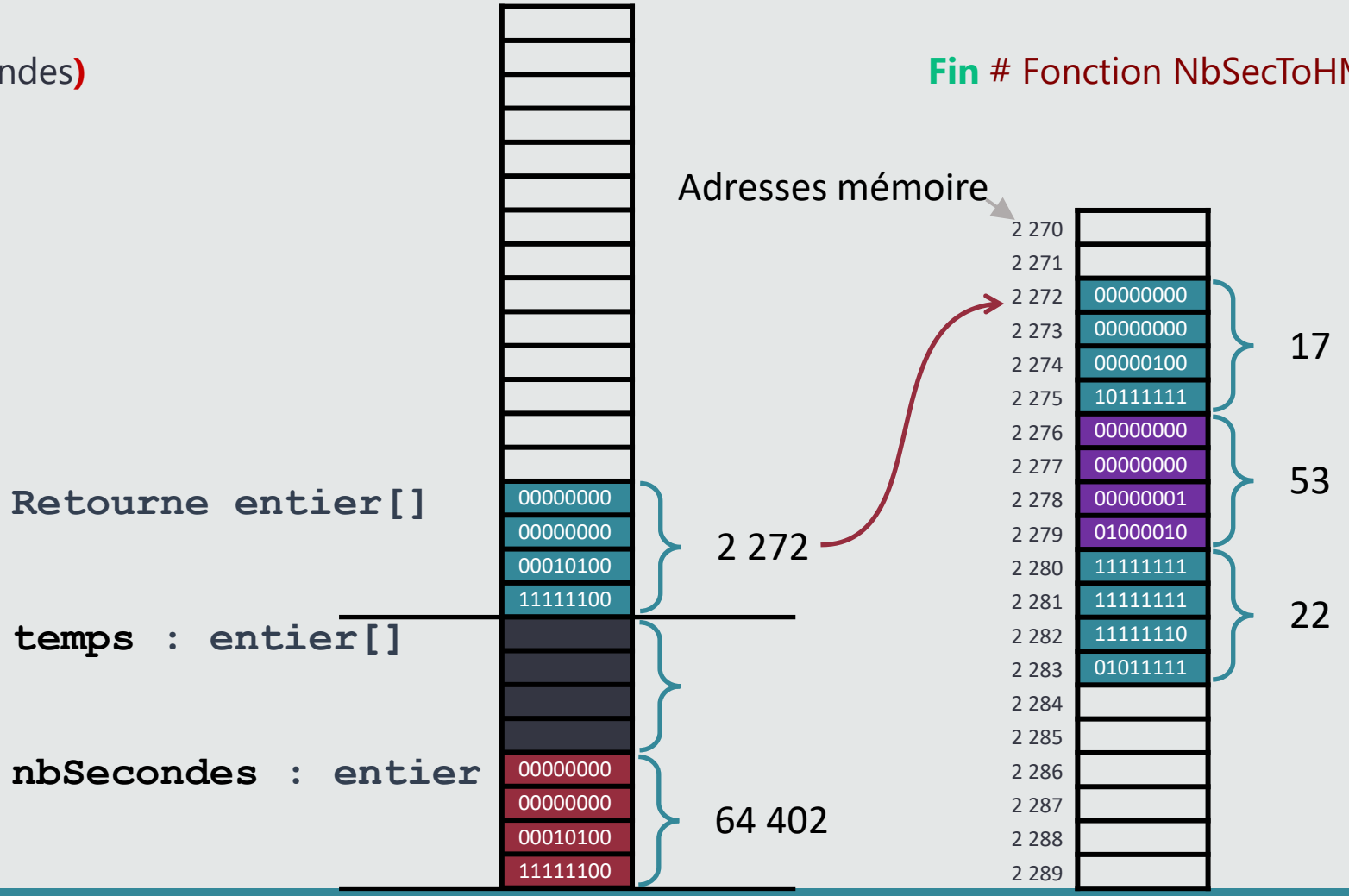
temps <- nbSecToHMS(nbSecondes)



Retourner un tableau

temps <- nbSecToHMS(nbSecondes)

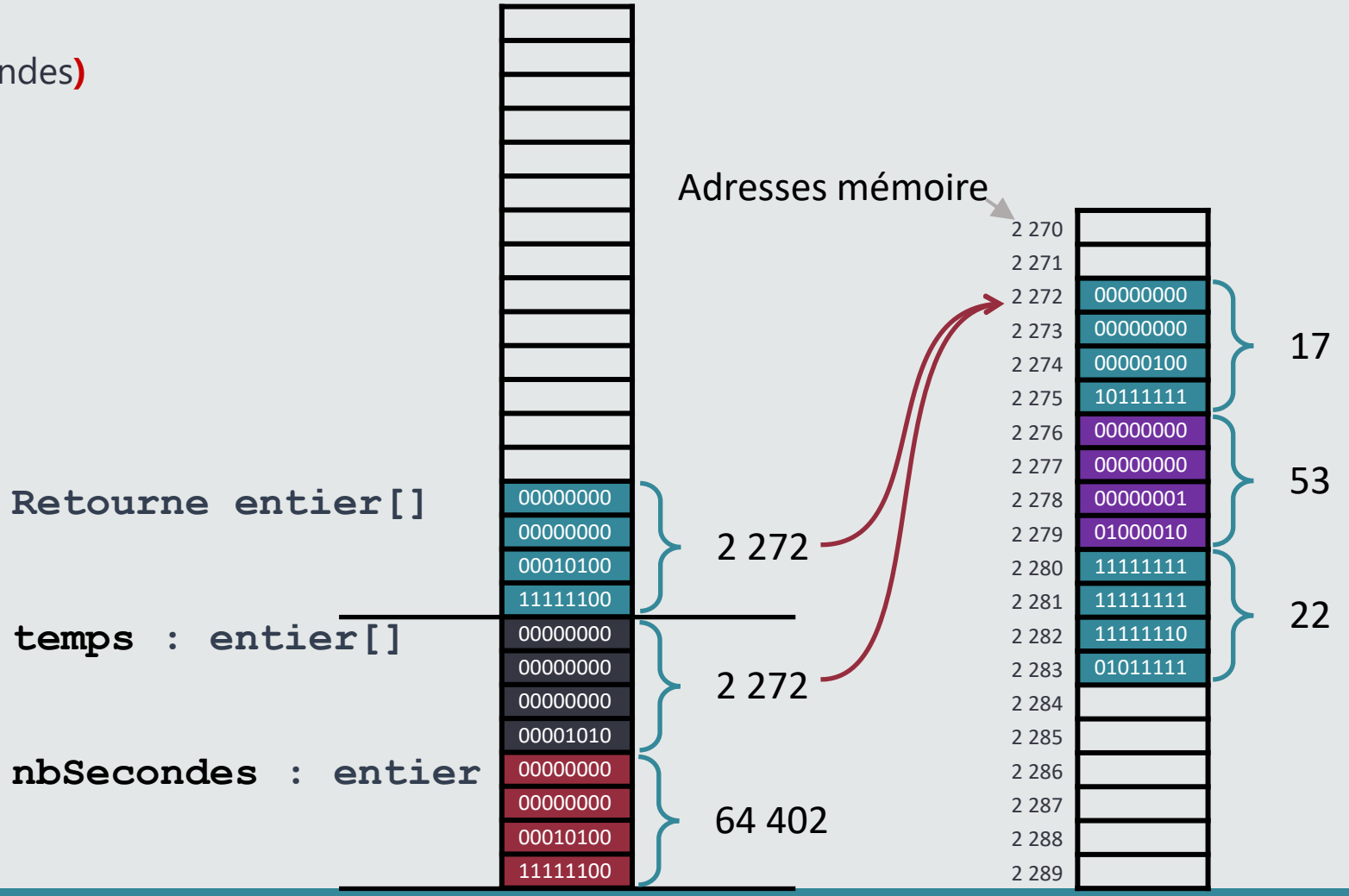
Fin # Fonction NbSecToHMS



Les procédures et fonctions

Retourner un tableau

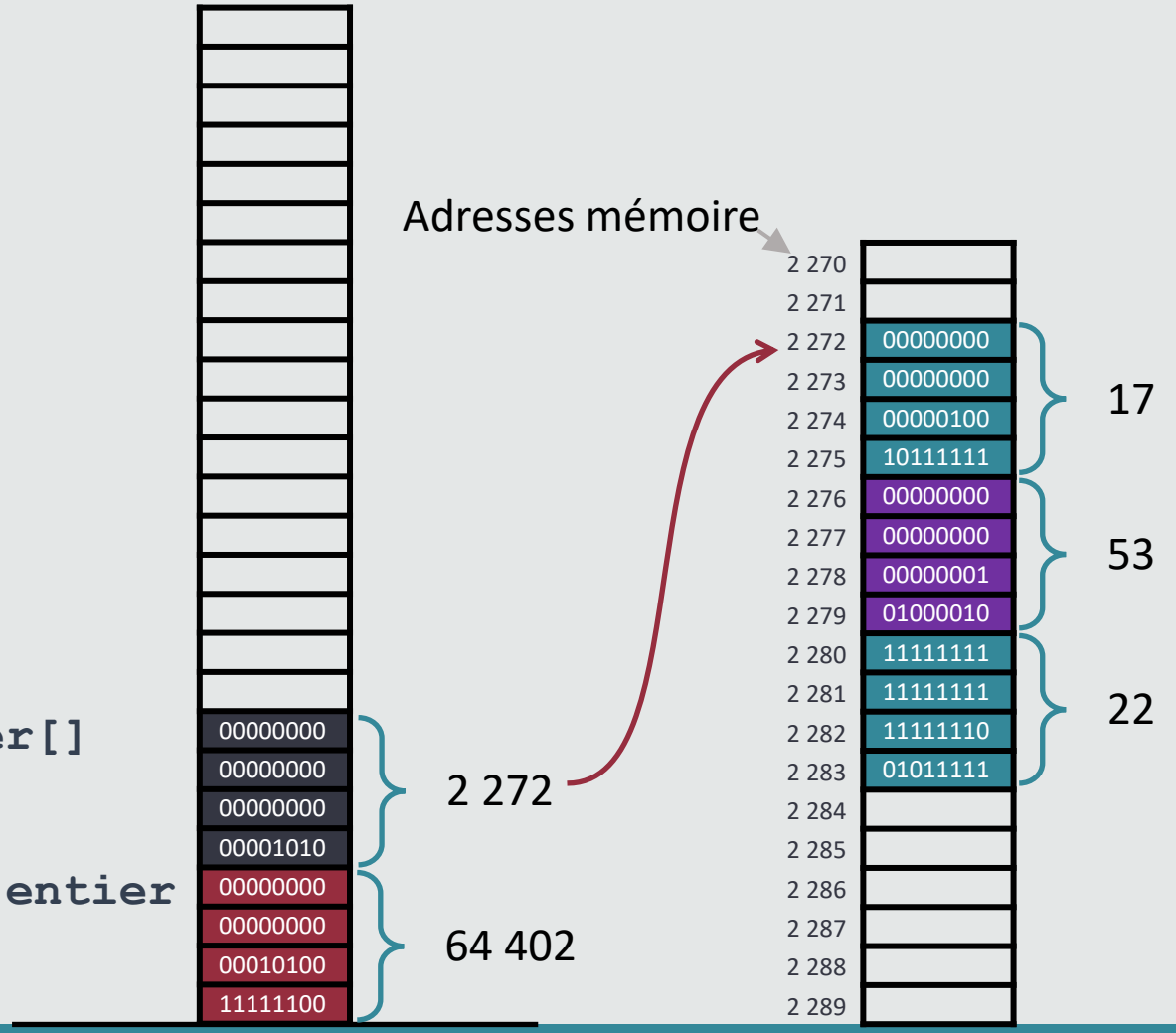
temps <- nbSecToHMS(nbSecondes)



Retourner un tableau

```
écrire("durée = " & temps[H] & ":" &
                    temps[M] & ":" & temps[S])
```

durée = 17:53:22



TD : Un tableau et des fonctions

- Écrire une fonction qui crée un tableau (à 1 dimension) d'entiers (TAILLE constante = 10) et initialise aléatoirement les valeurs de ce tableau avec des valeurs comprises entre une borne minimale et une borne maximale passées en paramètres

- Créer une fonction qui retourne la plus grande valeur d'un tableau

- Écrire un algorithme faisant appel à ces deux fonctions

TD : Un tableau et des fonctions

```
Constante TAILLE : entier <- 10

Fonction initialTabAleatoire(min : entier, max : entier) Retourne entier[]
# retourne un tableau de taille 'TAILLE' avec ses éléments initialisés
# avec des valeurs aléatoires comprises entre 'min' et 'max'.
Variable  tableau : entier[TAILLE]
Variable  i : entier
Début
    Pour i <- 0 à TAILLE - 1
        tableau[i] <- aléa(min, max)
    FPour
    Retourner tableau
Fin
```

TD : Un tableau et des fonctions

```
Fonction max(tableau : entier[], taille : entier) Retourne entier
# retourne la valeur maximale du tableau
Variable  max : entier <- tableau[0]
Variable  i : entier
Début
    Pour i <- 1 à taille - 1
        Si tableau[i] > max alors
            max <- tableau[i]
        FSi
    FPour
    Retourner max
Fin
```

TD : Un tableau et des fonctions

```
Algo tableauEtFonctions
# crée un tableau initialisé aléatoirement et affiche le maximum
Constante MIN : entier <- 1
Constante MAX : entier <- 100
Variable  tableau : entier[]
Début
    tableau <- initialTabAleatoire(MIN, MAX)
    écrire("Le maximum du tableau est " & max(tableau, TAILLE))
Fin
```

Algorithmique

Module 7 – TD Final

Objectifs

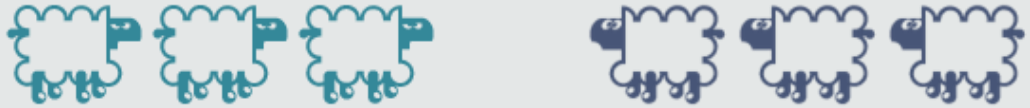
- Réutiliser l'ensembles des concepts vus précédemment
- Établir un plan stratégique de résolution d'un exercice d'algorithmique plus complexe que ceux vus antérieurement

TD Final

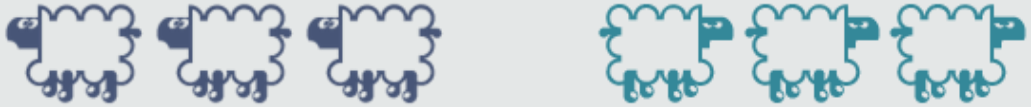
TD : jeu du saute-moutons

[Y. Secq - université Lille 1]

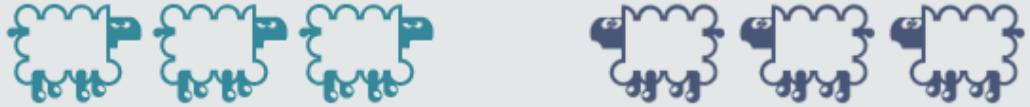
- Objectif : faire se croiser deux troupeaux de moutons
  - Position initiale :



  - Position finale :



- Règles
  - Un mouton ne peut pas reculer
  - Un mouton ne peut avancer d'un cran que si la place est libre



  - Un mouton peut sauter un mouton qui va en sens inverse si la place suivante est libre

TD Final

TD : Jeu du saute-moutons

- Représentation des données

	0	1	2	3	4	5	6
tableau	>	>	>		<	<	<

- Affichages et *saisies*

```
 0 1 2 3 4 5 6
|>|>|>|  |<|<|<|
Quel pion voulez vous déplacer?
2
 0 1 2 3 4 5 6
|>|>|  |>|<|<|<|
```

TD Final

TD : Jeu du saute-moutons

- Plan d'action
  - Essayer de jouer (avec un crayon et un papier)
    - Quelles sont les situations de blocage ?
    - Quels sont les différents déplacements possibles ?
  - Quels sous-algorithmes allez-vous créer ?
    - Sont-ils des procédures ou des fonctions ?
  - Écrire l'algorithme principal
  - Écrire les fonctions et procédures