La programmation en langage C

Chapitre 2

Andry RASOANAIVO

- **Alogorithmes et programmation**
- **Introduction au langage C**

Algorithmes et programmation

- **Définition**:un algorithme est la description d'une suite **d'actions** (écrites dans un langage plus ou moins proche d'une langue naturelle: Français) à effectuer, dans un **ordre donné**, pour parvenir à un résultat.
- **Définition**: Un programme est un algorithme traduit dans un langage de programmation (par exemple en C).

Mise au point d'un programme

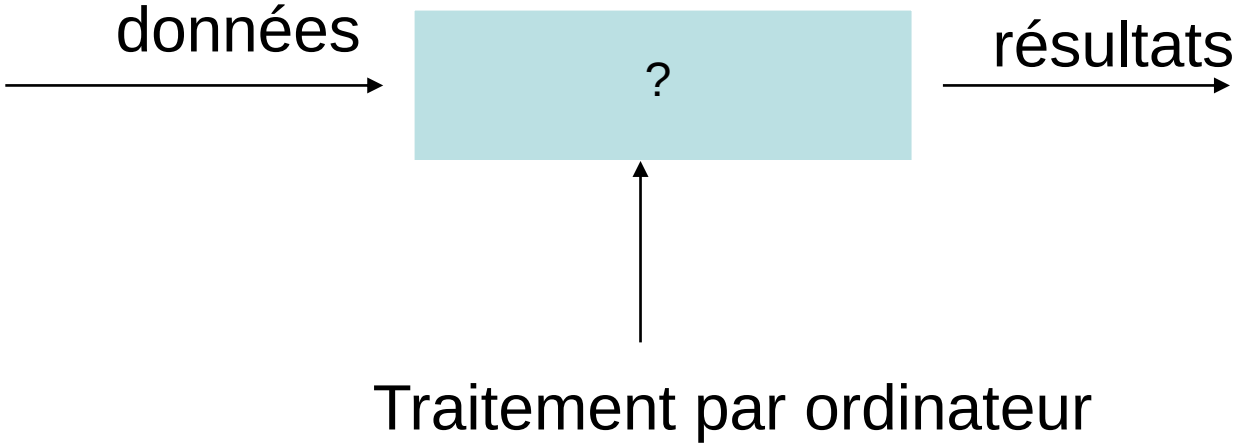
Un ordinateur est une machine capable d'exécuter des opérations, données par l'homme, à une très grande vitesse. Ces opérations doivent telles qu'elles doivent être comprises par l'ordinateur.

Pour réaliser le programme, correspondant à un

la résolution d'un certain problème, on devra indiquer à l'ordinateur **toutes les opérations à effectuer dans tous les cas que l'on juge possibles.**

La préparation d'un problème, en vue de son passage sur ordinateur va nécessiter plusieures étapes:

1. **Définition exacte du probème**: On ne peut pas espérer resoudre, à l'aide d'un calculateur, un problème non complétement défini. Il faudra en particulier définir clairement les **objectifs** que l'on souhaite atteindre (et leurs limitations éventuelles), à partir des **données** existantes.


données
?
résultats
Traitement par ordinateur

2. Choix de la méthode de résolution: Pour atteindre nos objectifs à partir des données existantes, une méthode résolution s'impose.

3. Étude détaillée de l´algorithme:

Une fois la méthode de résolution fixée, il reste à détailler l'algorithme correspondant, en étudiant tous les cas possibles qui peuvent se présenter afin que cette méthode puisse engendrer un résultat

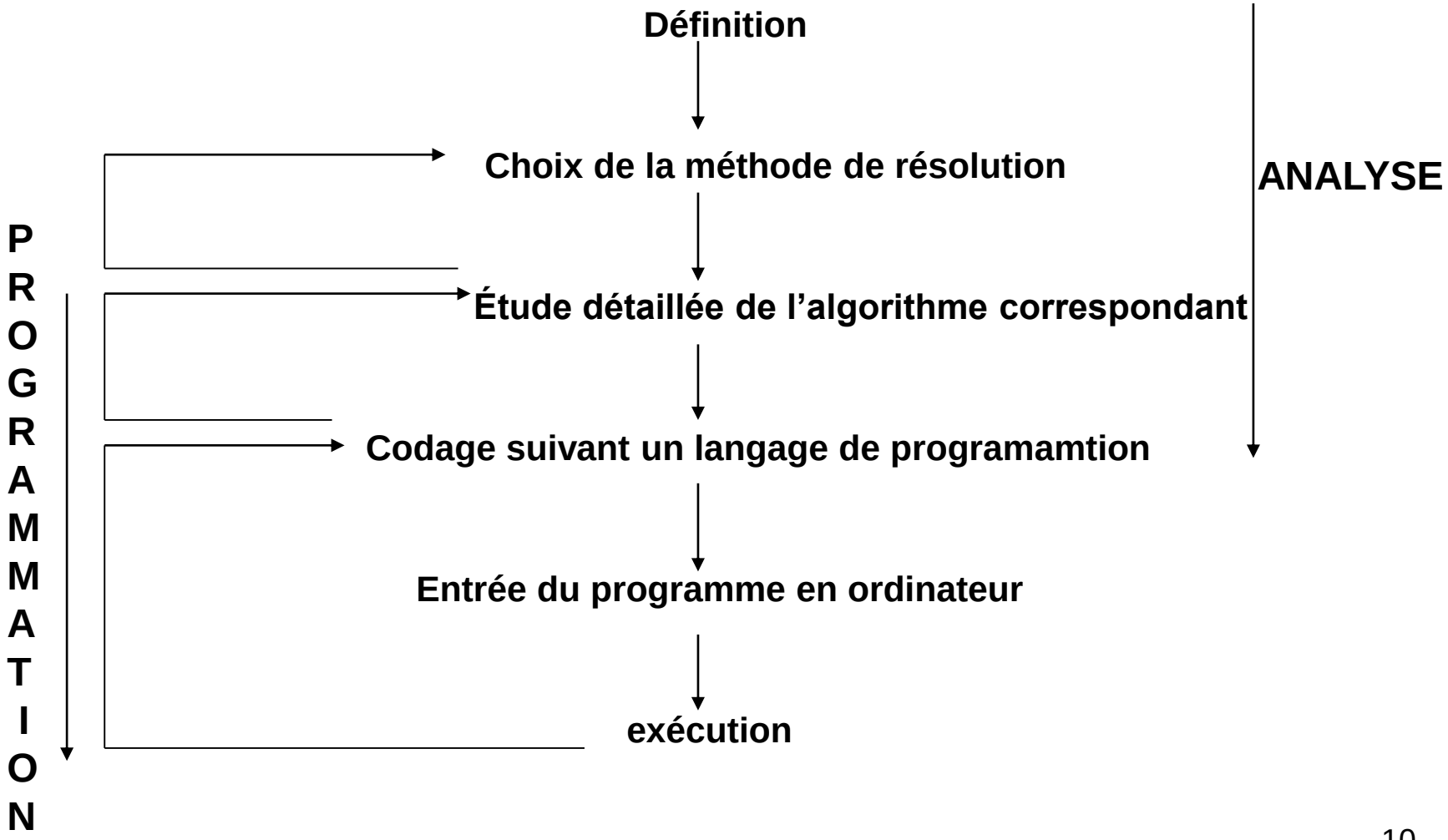
3. Codage et mise au point du programme:
L'algorithme étant explicité, il ne reste plus qu'à le traduire dans un langage de programmation, et à le rentrer dans la mémoire de l'ordinateur. Pour que ce programme puisse nous donner des résultats, deux étapes doivent suivre:

a. **Compilation:** traduction du programme dans le langage machine. Des erreurs peuvent survenir (lexicales, syntaxiques).

b. **Exécution:** une fois que la compilation est correcte, la phase d'exécution peut commencer. Le résultat obtenu est relatif aux données introduites.

- **erreurs:** le programme peut ne pas donner les résultats escomptées: revoir la méthode de résolution, l'algorithme ou le codage.

c. **documentation:** mode d'emploi, fonction, restrictions, ... Pour être vraiment utilisable.


Définition
Choix de la méthode de résolution
Étude détaillée de l'algorithme correspondant
Codage suivant un langage de programamtion
Entrée du programme en ordinateur
exécution
ANALYSE
PROGRAMMATION

Déterminer la moyenne de 4 nombres positifs

1. **Définition du problème**

   données: 4 nombres

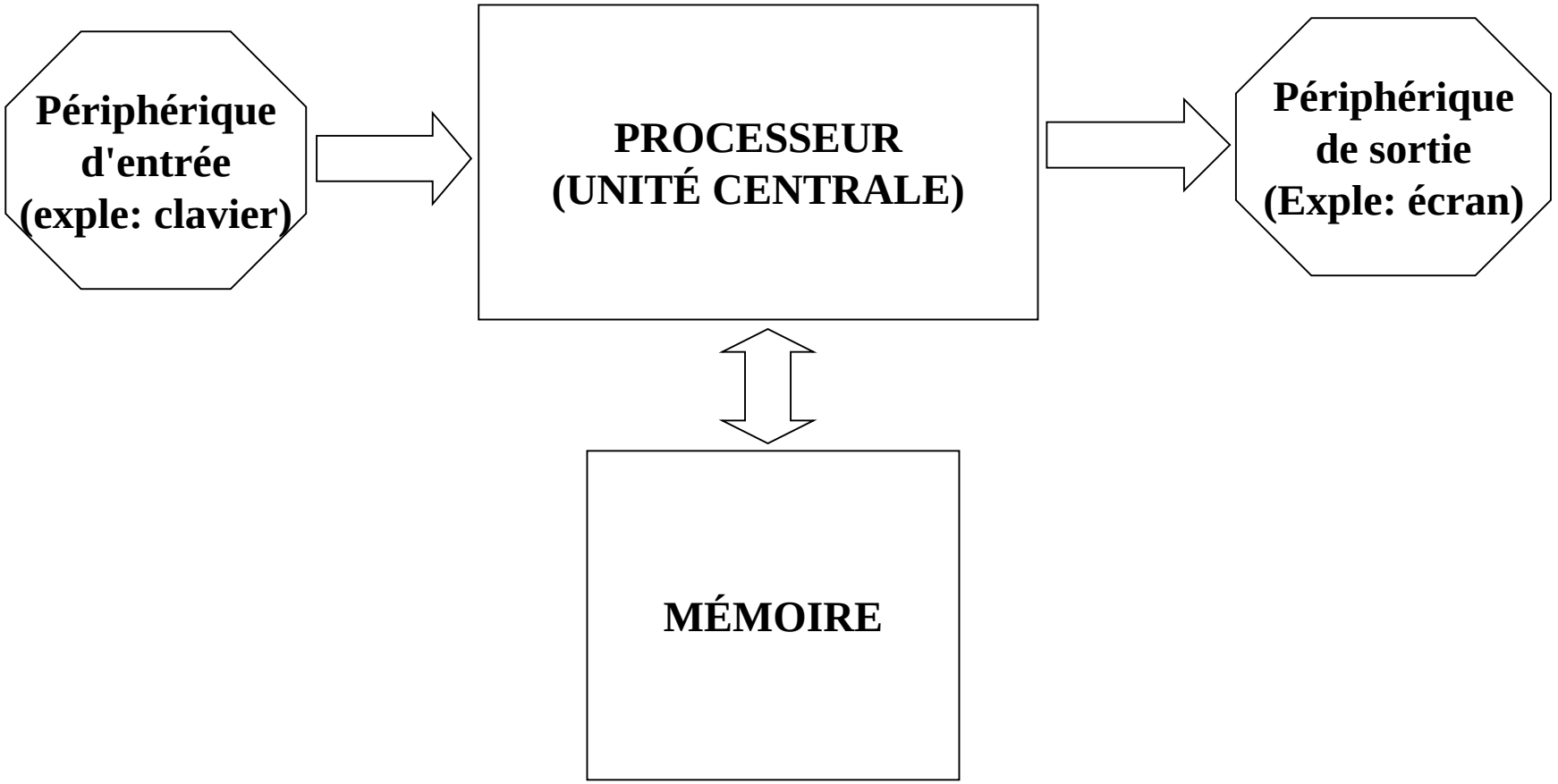
   objectifs: la moyenne en sortie

2. **Choix de la méthode**

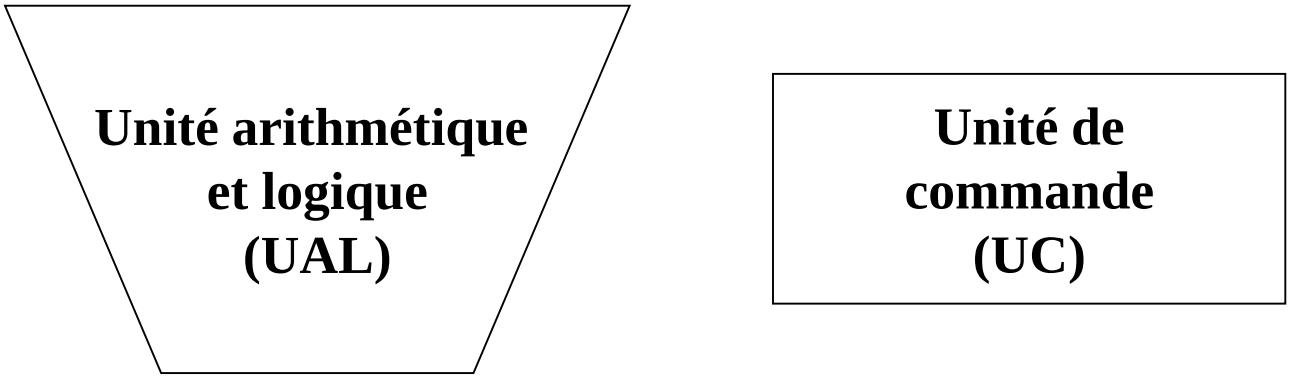
Additionner tous les nombres et faire la division par 4.

- **Algorithme:** Avant de traduire l’algorithme, correspondant à cette méthode de résolution, il est très utile de faire quelques rappels:

Structure d’un ordinateur



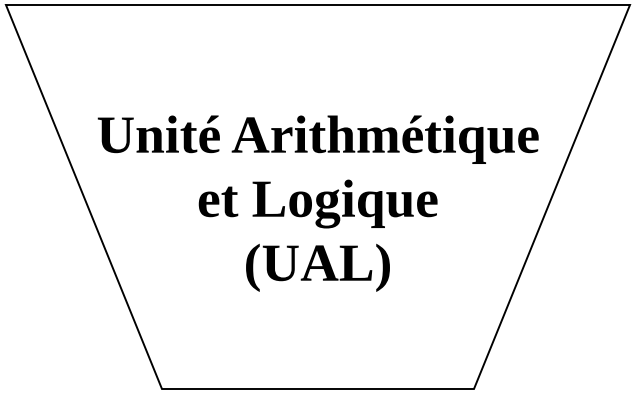
Unité centrale



UAL: Sert à effectuer les opérations arithmétiques, les comparaisons et les opérations logiques

UC: Contrôle les communications entre l'UAL, la mémoire, et les périphériques

Quelles opérations arithmétiques peuvent-elles être réalisées par l'UAL du processeur?



Seulement 4 opérations:

+	addition
-	soustraction
*	multiplication
/	division

Les expressions

À partir des 4 opérations de base il est possible de construire différentes expressions.

Une expression est une formule arithmétique pouvant être évaluée par le processeur.

Exemple:

- **2 + 3 + 4 = 9**
- **2 + (3 + 4) = 9**
- **(2 + 3) *4 = 20**
- **2 + (3 * 4) = 14**
- **2 + 3 * 4 = ?**

Question: Comment le processeur évalue-il la dernière expression?

Priorité des opérateurs

Priorité maximale	( )	**Les expressions entre parenthèses sont évaluées en premier.**
Priorité moyenne	* /	**La multiplication et la division ont la même priorité.**
Priorité minimale	+ -	**L'addition et la soustraction ont la même priorité.**

Exemples de priorité des opérateurs

2 + 3 * 4 / 2 * (3 * 5) - 7

On évalue d'abord les parenthèses

2 + 3 * 4 / 2 * 15 - 7

***** et / ont une priorité supérieure à + et -

On les évalue de gauche à droite

2 + 12 / 2 * 15 - 7

2 + 6 * 15 - 7

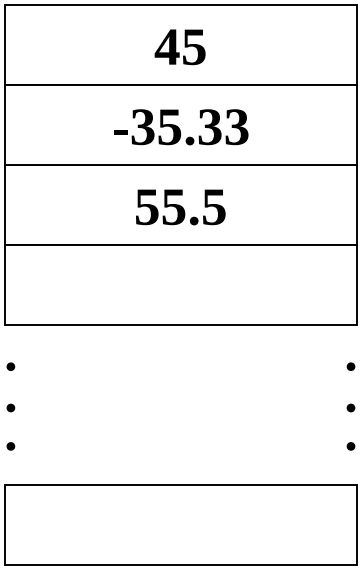
2 + 90 - 7

+ et - ont la même priorité et sont évalués de gauche à droite

92 - 7

85

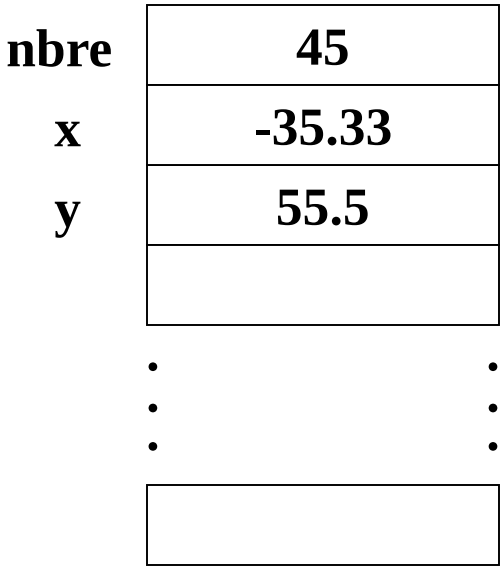
La mémoire



Le processeur tient sa puissance de la mémoire avec laquelle il communique constamment.

La mémoire peut être vue comme une boîte dont chaque case peut contenir une valeur numérique.

Les variables

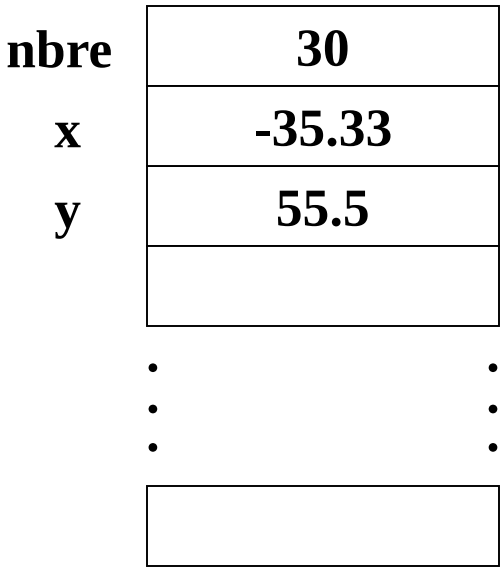


Afin d'identifier clairement où se trouve une valeur donnée, il est nécessaire de donner des noms aux cases.

Par exemple, la case **nbre** contient la valeur 45, la case **x** contient la valeur **-** 35.33 et la case **y** contient la valeur 55.5.

Ces noms sont appelés **variables** car le contenu de la case mémoire associée peut varier au cours de l'exécution d'un programme.

Les variables et les expressions



Le processeur est capable d'évaluer des expressions contenant des variables en remplaçant celles-ci par leur valeur respective.

Exemple

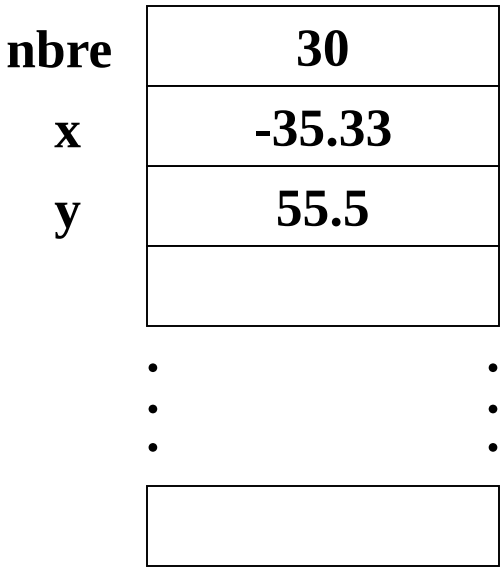
x+y + 2 * nbre

-35.33 +55.5+ 2 * 30

20.17 + 60

80.17

Les différents types de variables



On remarque que certaines cases contiennent des valeurs entières alors que d'autres contiennent des valeurs réelles.

Il est important d'indiquer au processeur quel type de valeurs contient chaque case utilisée.

Cela est nécessaire puisque le processeur traite différemment les entiers et les réels (et d'autres types de données, tels que les caractères, les données logiques (booléens).

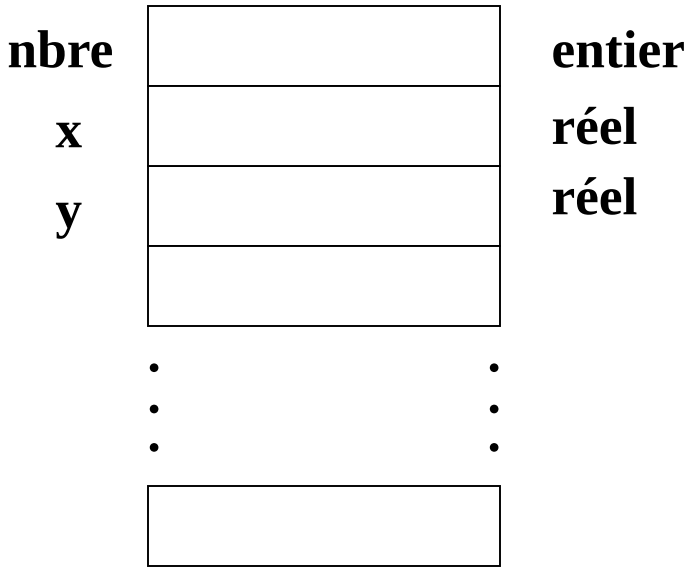
Les déclarations

Les déclarations servent à donner un nom à une case mémoire et à indiquer le type de valeurs qu'elle peut contenir.

Une déclaration typique ressemble à:

entier nbre
réel x,y

Après une telle déclaration, le processeur configure la mémoire comme dans l' illustration ci-contre.



Remarques concernant les déclarations

Remarques concernant les déclarations

Lorsque l'on déclare une variable, on ne lui donne pas de valeur.
On ne fait qu' indiquer au processeur le nom et le type des variables.

Remarques concernant les déclarations

Lorsque l'on déclare une variable, on ne lui donne pas de valeur.
On ne fait qu' indiquer au processeur le nom et le type des variables.

Le choix des noms de variables est arbitraire mais devrait toujours refléter ce que représente les valeurs qu'elles contiennent.

Remarques concernant les déclarations

Lorsque l'on déclare une variable, on ne lui donne pas de valeur.
On ne fait qu' indiquer au processeur le nom et le type des variables.

Le choix des noms de variables est arbitraire mais devrait toujours refléter ce que représente les valeurs qu ’elles contiennent.

Le contenu d'une case mémoire peut être modifié au cours de l'exécution d'un programme mais pas le type de valeur qu'elle contient.

Remarques concernant les déclarations

Lorsque l'on déclare une variable, on ne lui donne pas de valeur.
On ne fait qu' indiquer au processeur le nom et le type des variables.

Le choix des noms de variables est arbitraire mais devrait toujours refléter ce que représente les valeurs qu ’elles contiennent.

Le contenu d'une case mémoire peut être modifié au cours de l'exécution d ’un programme mais pas le type de valeur qu'elle contient.

Les déclaration sont faite à l' intérieur d'un programme donné.
Lorsque le programme se termine le nom, le type et la valeur contenue dans les cases mémoire disparaisse.

Instruction d’affectation

- Pour mettre une valeur dans une variable, on utilise une **instruction d'assignation**.

- Une affectation a la forme suivante:

variable ← expression

- L'expression est d'abord évaluée. Ensuite, le résultat est mis dans la case mémoire correspondant à la variable

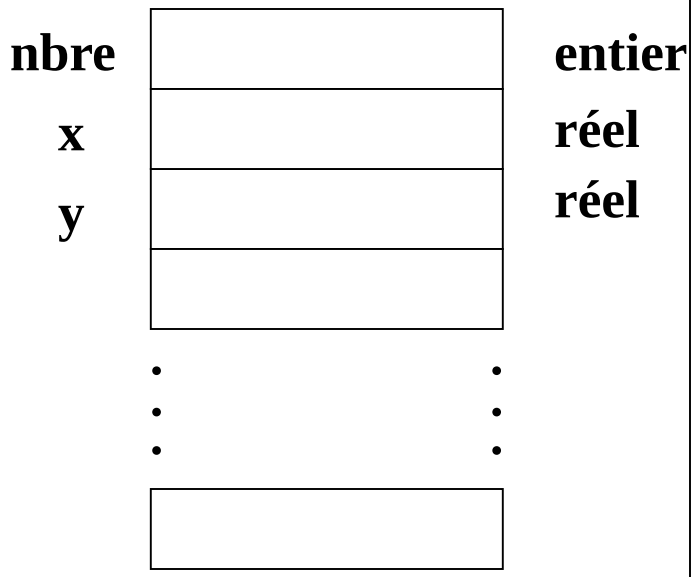
Exemple d'affectation

Après la déclaration

entier nbre

réel x,y

la mémoire ressemble à la figure suivante:



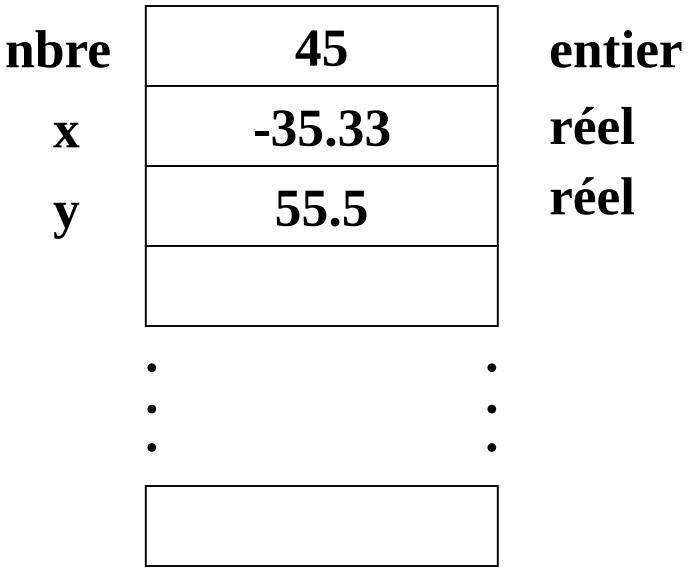
Si par la suite le processeur exécute les instructions suivantes:

nbre ← 45

x ← -35.33

y ← 55.5

La mémoire devient:



Algorithme calculant la moyenne

La séquence d'instructions nécessaire au calcul de la moyenne est donc:

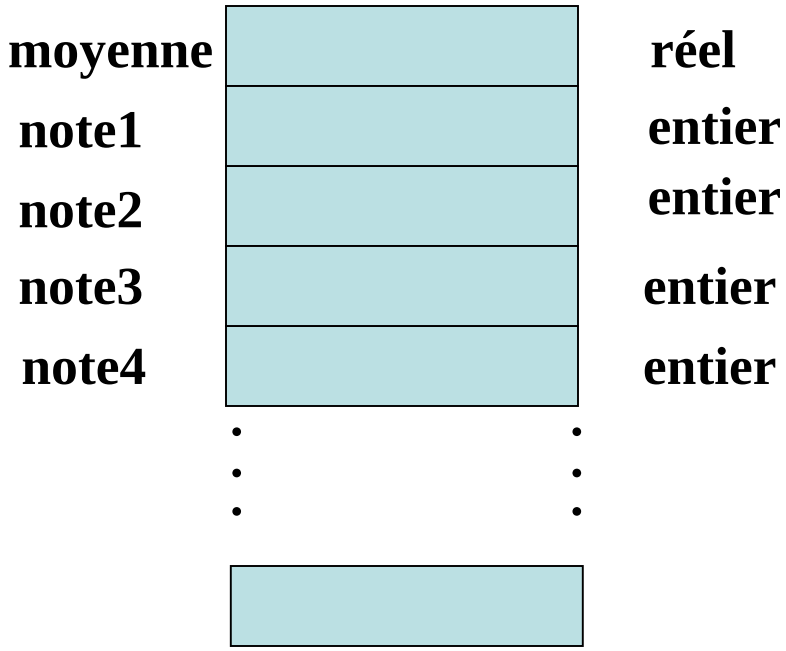
```
réel moyenne
entier note1, note2, note3, note4

note1 ← 75
note2 ← 100
note3 ← 88
note4 ← 95

moyenne ← note1 + note2 + note3 + note4   / 4

ecrire moyenne
```

Remarque: Les instructions sont toujours exécutées de façon séquentielle par le processeur.

Après les déclarations suivantes,

réel moyenne
entier note1, note2, note3, note4

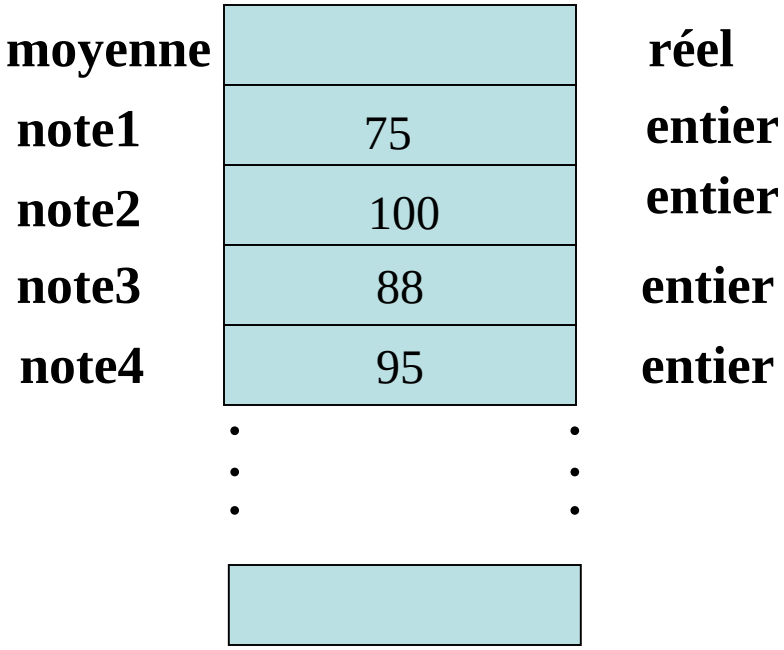
la mémoire ressemble à la figure ci-contre



Après les instructions d'affectation suivantes:

note1 ← *75*
note2 ← *100*
note3 ← *88*
note4 ← *95*

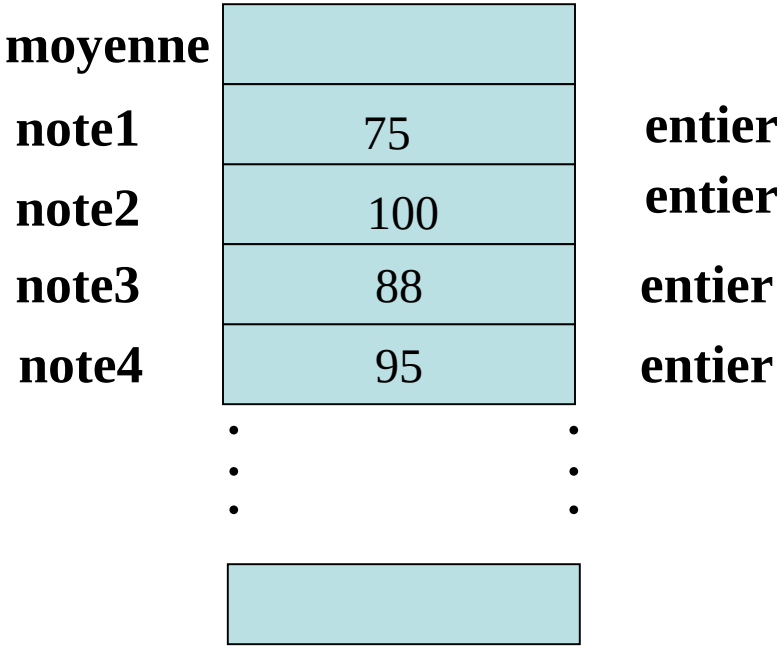
La mémoire devient:



Remarque: la variable **moyenne** est toujours indéfinie puisqu'aucune valeur n'y a été affectée.

Pour calculer la moyenne de ces 4 valeurs et mettre le résultat dans la variable **Moyenne**, le processeur doit exécuter l' instruction suivante:

moyenne ← (note1 + note2 + note3 + note4) / 4



Le processeur évalue d'abord l'expression de gauche:

moyenne ← (note1 + note2 + note3 + note4) / 4

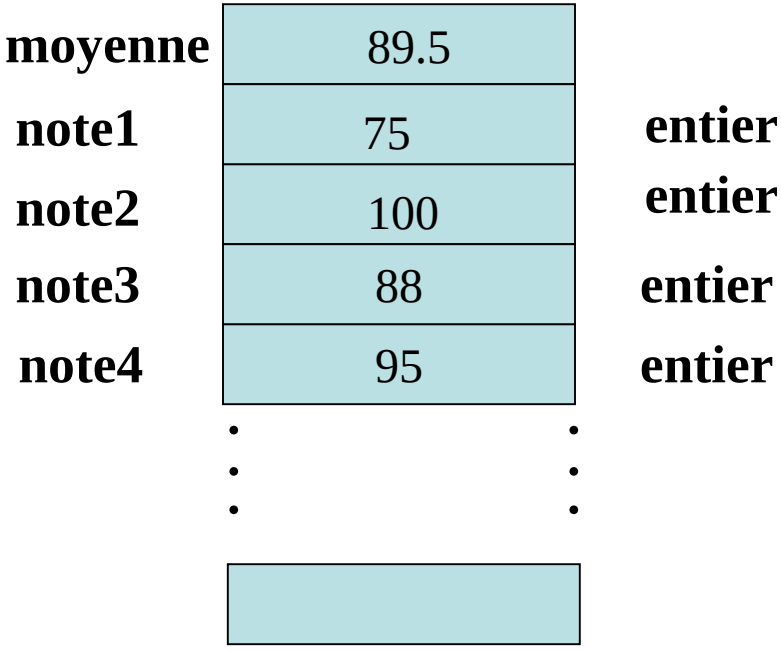
moyenne ← (75 + 100 + 88 + 95) / 4

moyenne ← (175 + 88 + 95) / 4

moyenne ← (263 + 95) / 4

moyenne ← 358 / 4

moyenne ← 89.5



Le résultat est ensuite déposé dans la case mémoire associée à la variable **moyenne.**

Que se passerait-il si on omettait les parenthèses dans l'expression précédente?

moyenne ← note1 + note2 + note3 + note4 / 4

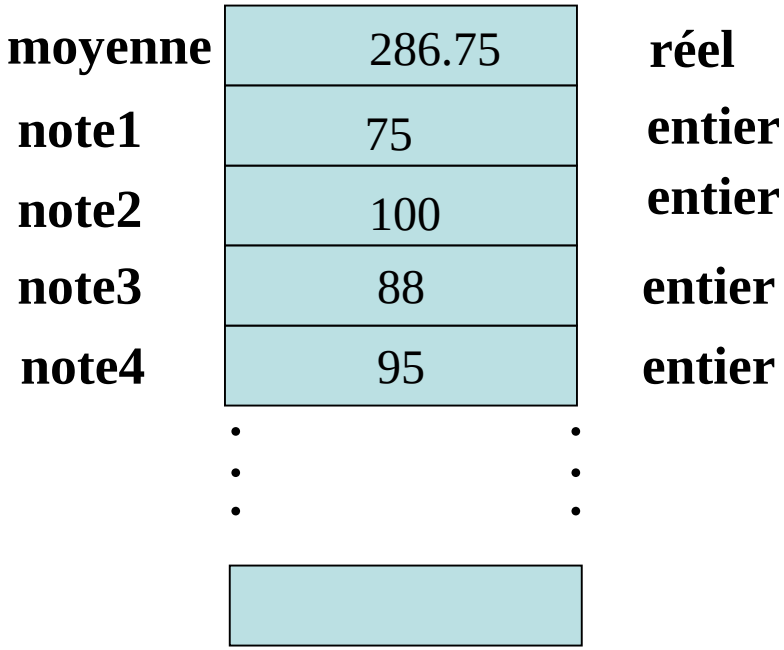
moyenne ← 75 + 100 + 88 + 95 / 4

moyenne ← 75 + 100 + 88 + 23.75

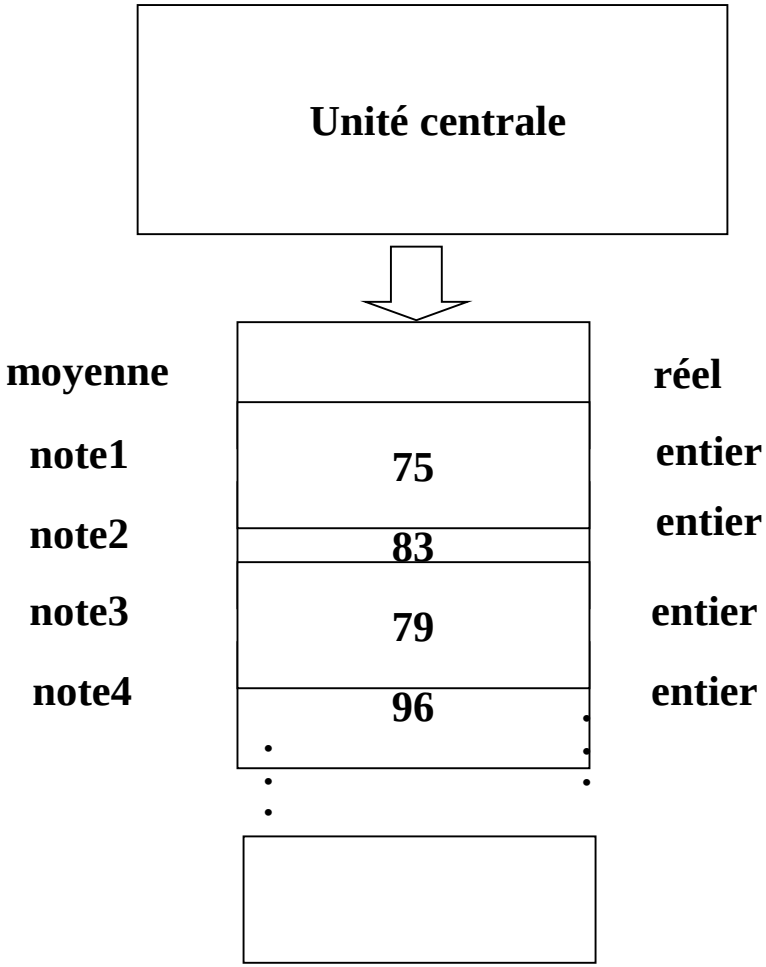
moyenne ← 175 + 88 + 23.75

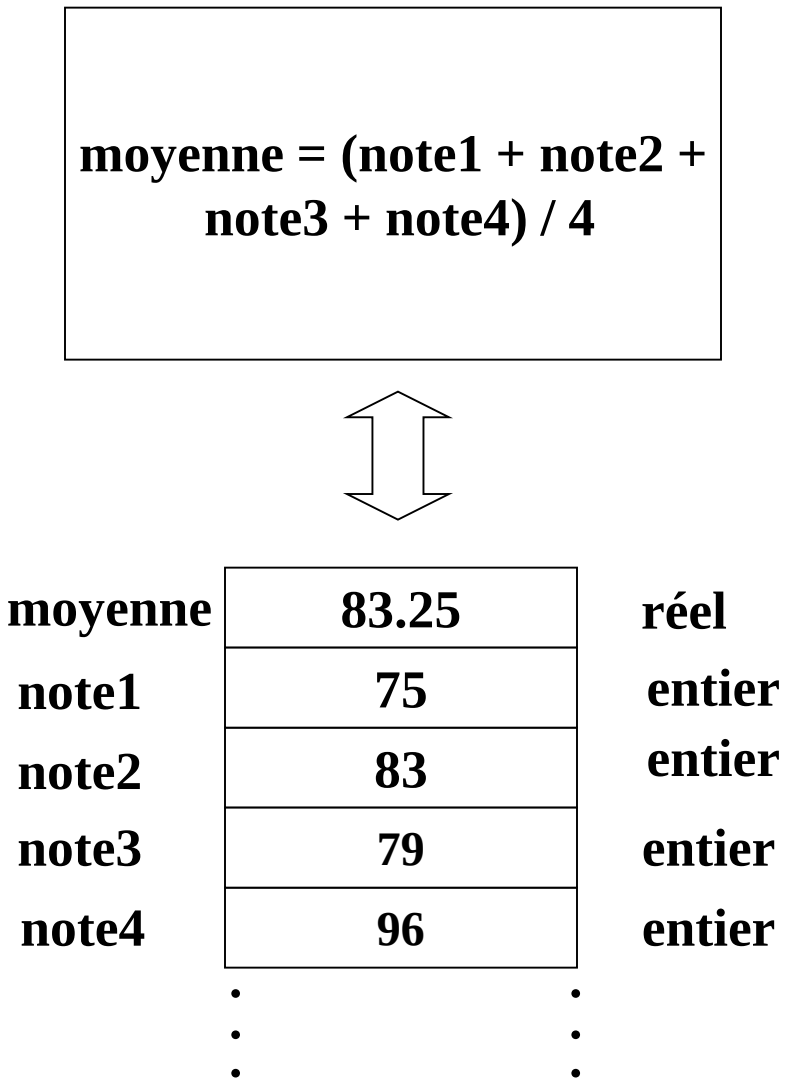
moyenne ← 263 + 23.75

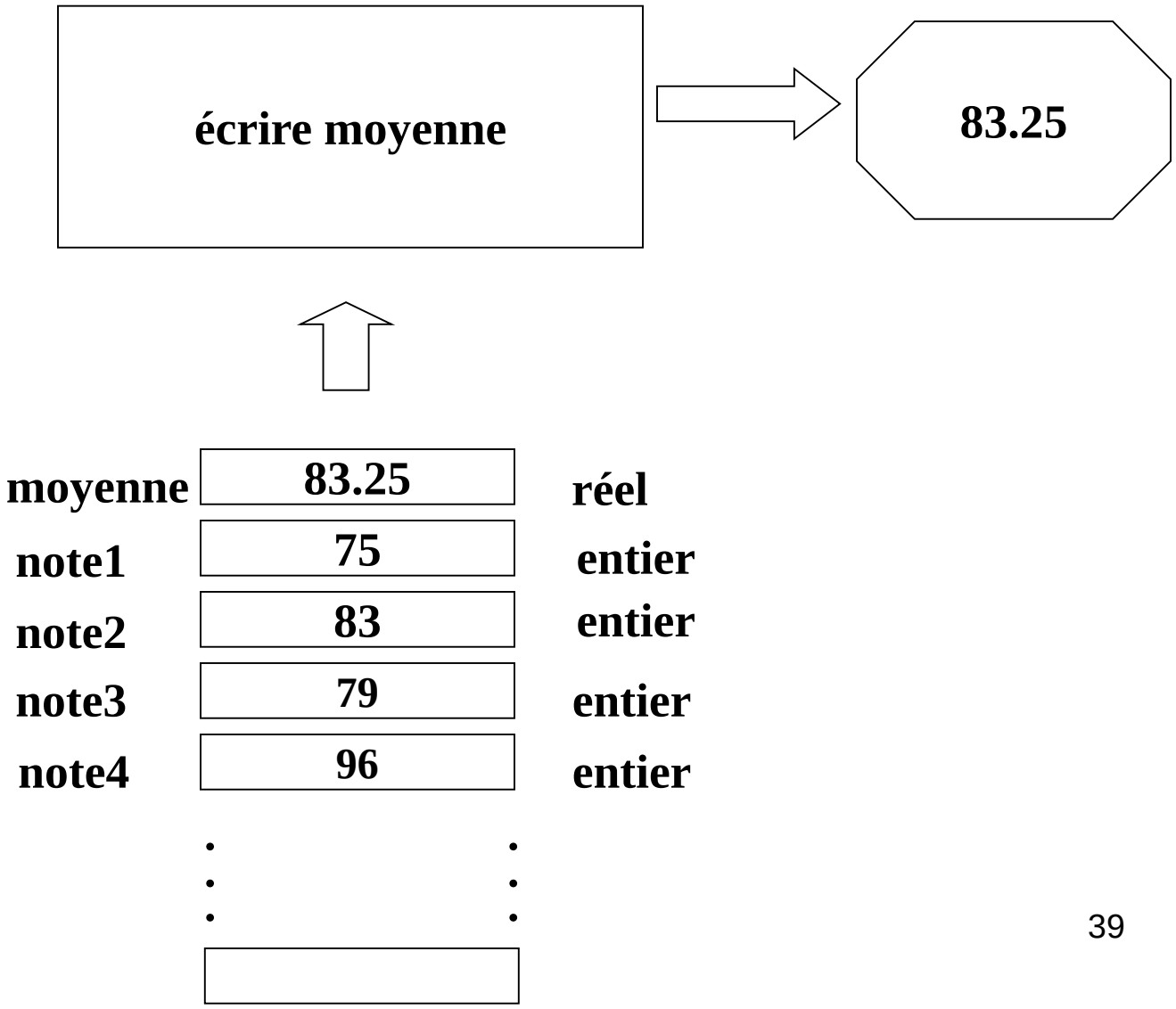
moyenne ← 286.75



Le mauvais résultat serait ensuite déposé dans la case mémoire associée à la variable **moyenne**.


Unité centrale
moyenne
réel
note1
75
entier
note2
83
entier
note3
79
entier
note4
96
entier


moyenne = (note1 + note2 + note3 + note4) / 4
moyenne
83.25
réel
note1
75
entier
note2
83
entier
note3
79
entier
note4
96
entier


écrire moyenne
83.25
moyenne
83.25
réel
note1
75
entier
note2
83
entier
note3
79
entier
note4
96
entier

- **Remarque:** cet algorithme fonctionne **seulement** pour les 4 valeurs particulières 75, 83,79 et 96.

- Si l'on désire (ce qui est mieux car plus général) que cet algorithme fonctionne pour plusieurs valeurs, on aura besoin de les introduire par un périphérique d'entrée (par exemple le clavier).

```
réel moyenne
entier note1, note2, note3, note4

note1 ← 75
note2 ← 83
note3 ← 79
note4 ← 96

moyenne ← note1 + note2 + note3 + note4   / 4
ecrire moyenne
```

Ne calcule la moyenne que de ces 4 valeurs particulières

```
réel moyenne
entier note1, note2, note3, note4

lire note1 note2 note3 note4

moyenne ← note1 + note2 + note3 + note4   / 4

ecrire moyenne
```

Calcule la moyenne de n'importe quelle valeurs

Introduction au langage C

Introduction

- Nous avons vu un modèle simple d'ordinateur et nous avons discuté des principaux éléments du ***langage*** utilisé pour communiquer avec lui: les déclarations, les expressions, les instructions d'affectation, de lecture et d'écriture.

- Nous allons maintenant voir un **vrai langage de programmation**: **le langage C**. Nous commencerons par voir comment tous les éléments que nous avons vus peuvent être traduits en C.

Toutes les valeurs en C sont du type réel ou entier. En fait il s'agit de deux « familles » de types puisqu'il existe plusieurs sortes d'entiers et de réels. Voici les deux principaux types en C.

int : Les variables de ce type peuvent contenir un nombre entier, qui reflète typiquement la taille naturelle des nombres entiers sur la machine utilisée (32 bits sur sunchb).

double : Les variables de ce type peuvent contenir un nombre réel en point flottant en double précision (64 bits)

Les constantes

Les constantes de type int.

décimales:	17	8	-68	
octales:	021	010	-0104	(commence par un zéro)
hexadécimales:	0x11	0X8	-0x44	(commence par 0x ou 0X)

Les constantes de type double.

point flottant:	1200.0	3.1416	-.00067
avec exposant:	12e2	31416E-4	-67e-5
mixtes:	1.2e3	3.1416e0	-6.7E-4

Déclaration des variables

La déclaration des variables se fait selon le modèle suivant:

type variable1, variable2, ... ;

Remarquez la **virgule** séparant deux variables ainsi que le **point-virgule** à la fin.

Exemples:

int nbre; définit nbre comme une variable entière

int a,b,c; définit trois variables entières dont les noms respectifs sont a, b et c.

double x; définit x comme une variable réelle.

Le nom d'une variable peut contenir jusqu’à 256 caractères.

Les caractères permis sont

1) les lettres majuscules et minuscules (le C fait la différence entre les deux),
2) les chiffres
3) le caractère "_ "

Les noms ne peuvent pas commencer par un chiffre.

Une dernière restriction concernant les noms des variables: il est interdit d'utiliser un des 32 mots réservés du langage C

Mots réservés du langage C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Instruction d‘affectation (assignation)

L'instruction **variable ← expression** se traduit en C par

variable = expression ;

Remarque 1: Observez le caractère ' ; ' qui termine toujours une instruction en C.

Remarque 2: Le symbole **=** n'est pas un opérateur de comparaison en C. Il s'agit d'un opérateur d‘affectation.
Nous reviendrons sur ce sujet dans un prochain cours.

Les opérateurs arithmétiques

+ addition
- soustraction
*** multiplication**
/ division

engendre un int si les deux opérandes sont de type int, engendre un double sinon

% opérateur de modulo }

ne s'applique qu'à des opérandes de type int. Engendre un int.

donne le reste de la division entière

Exemple: 4 % 3 vaut 1
12 % 3 vaut 0
17 % 5 vaut 2

Les opérateurs de comparaisons

type d'opérateur	notation mathématique	notation en C
égal	=	==
plus grand	>	>
plus grand ou égal	≥	>=
plus petit	<	<
plus petit ou égal	≤	<=
différent	≠	!=

Les opérateurs logiques

Type d ’opérateur	Notation en C
ET	&&
OU	||
NON	!

La priorité des opérateurs en C

Opérateurs	Associativité
( )	de gauche à droite
!	de droite à gauche
* /	de gauche à droite
+ -	de gauche à droite
< <= > >=	de gauche à droite
== !=	de gauche à droite
&&	de gauche à droite
||	de gauche à droite
=	de droite à gauche

Les entrées

Après avoir déclaré des variables, il est possible d'y assigner des valeurs en provenance du péripherique d'entrée.

L'instruction

lire variable1 variable2 …

est exprimée en C de la façon suivante:
La fonction **printf()** et la fonction **scanf()**

scanf(“%type_variable1”,&variable1)

Les sorties

Écrire sur écran est très simple en C.

L'instruction :

ecrire expression1 expression2 …

s'exprime de la façon suivante

printf()

Exemple: Calcul de la moyenne

```
entier note1 note2 note3 note4
reel moyenne

lire note1 note2 note3 note4
moyenne ← (note1 + note2 + note3 + note4) / 4
ecrire moyenne
```

Faire l'écriture du programme en C?

Algorithme:

```
entier note1 note2 note3 note4
reel moyenne

lire note1 note2 note3 note4
moyenne ← (note1 + note2 + note3 + note4) / 4
ecrire moyenne
```

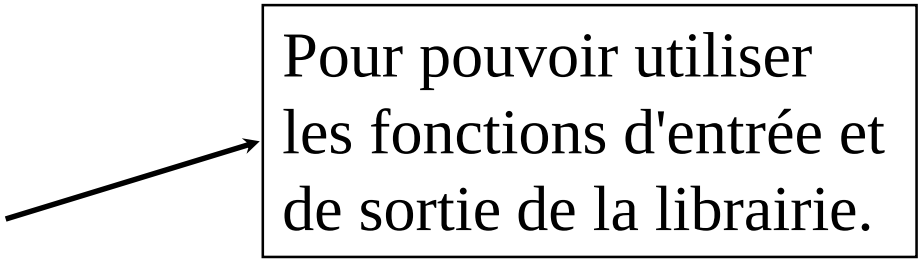
Traduction en C:

- Si on a les 4 notes suivantes: 2, 1, 1 et 1.

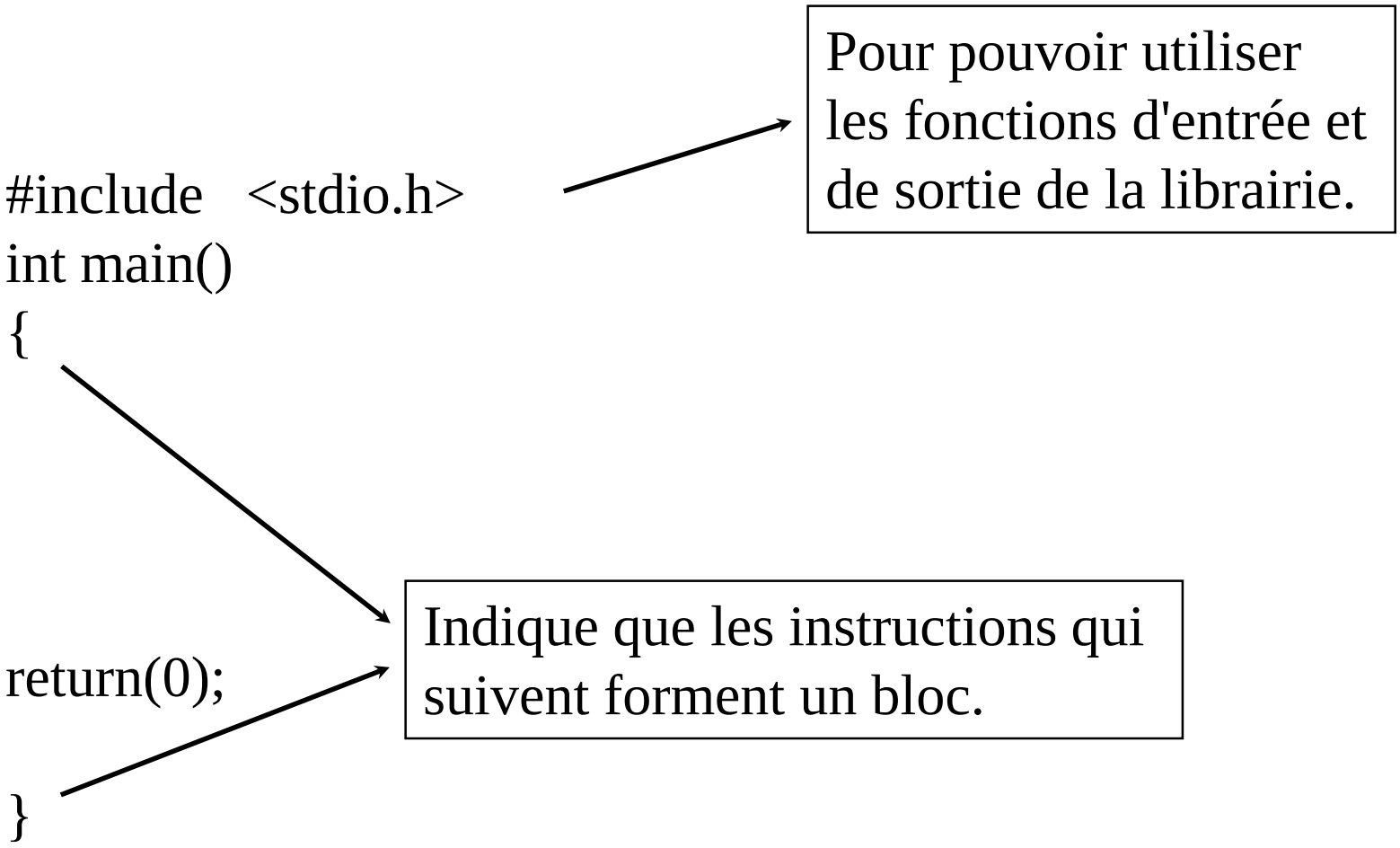
On a alors

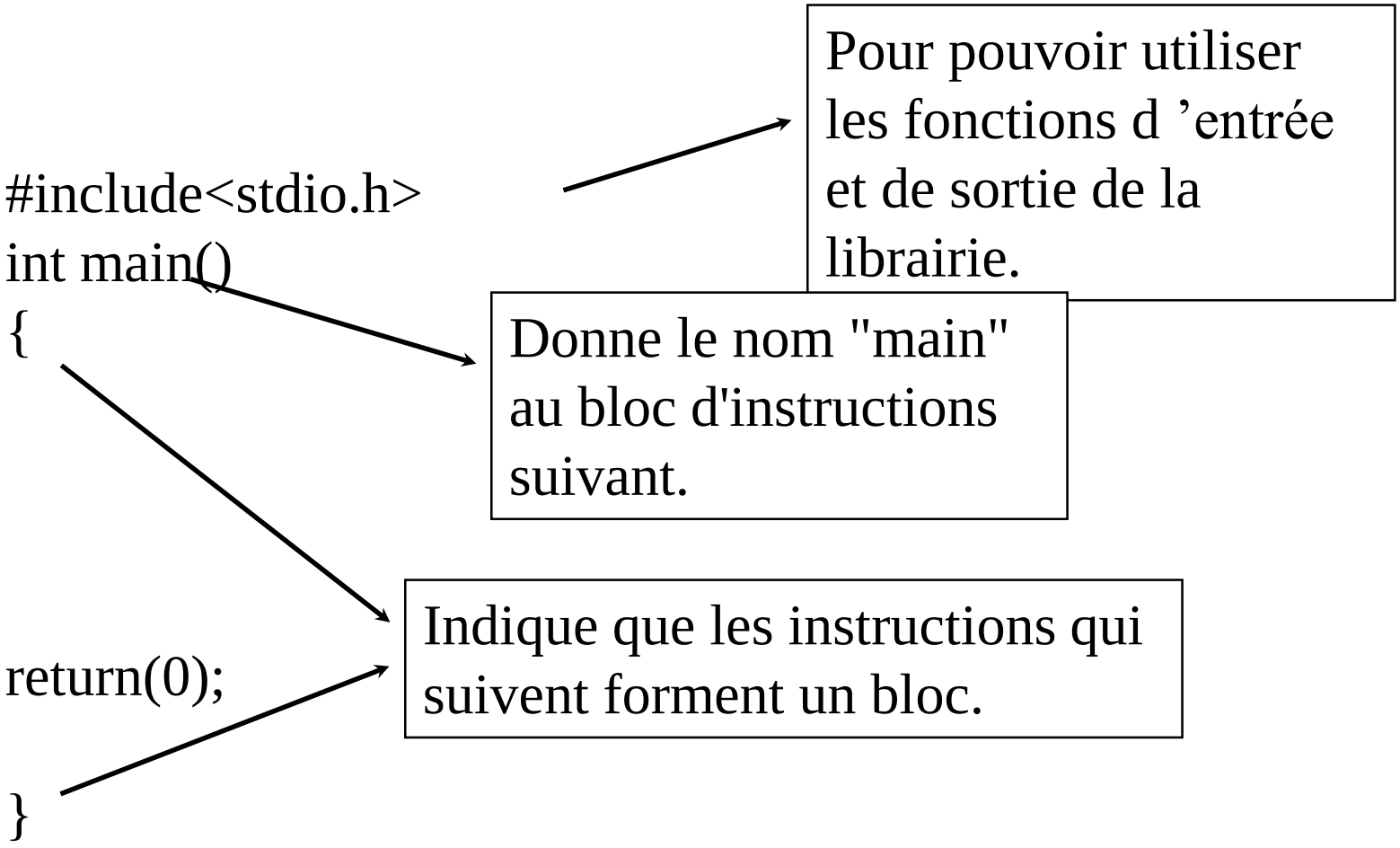
(2+1+1+1) / 4
= (3 +1 +1) / 4
= (4 + 1) / 4
= 5 / 4
= 1

Parce que si les deux opérandes de la division sont des entiers alors le résultat est un entier.




```
#include<stdio.h>
int main()
{




return(0);
}
```


#include <stdio.h>
int main()
{
return(0);
}
Pour pouvoir utiliser les fonctions d'entrée et de sortie de la librairie.
Indique que les instructions qui suivent forment un bloc.


#include<stdio.h>
int main()
{
return(0);
}
Pour pouvoir utiliser les fonctions d ’entrée et de sortie de la librairie.
Donne le nom "main" au bloc d'instructions suivant.
Indique que les instructions qui suivent forment un bloc.

Un autre exemple: Calcul des intérêts

- **Description du problème:** Étant donné un taux d'intérêts annuel et un montant à rembourser, afficher le montant mensuel à payer afin de rembourser capital et intérêts en 12 mois.
- **Entrée:** Deux nombre réels. Le premier est un taux d'intérêt (ex. 8.25 pour 8.25%) et le second est un montant à rembourser.
- **Sortie:** Un nombre réel représentant le paiement mensuel.

Après avoir demandé à votre banquier

T = Taux d'intérêts annuel

M = 1 + T/1200

P = Montant du prêt

Remboursement mensuel = $P \times (M - 1) \times \left(\frac{M^{12}}{M^{12} - 1} \right)$

Algorithme 1

réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

Tmp1 ⟵ 1 + taux_annuel / 1200

tmp2 ⟵ tmp1 **à la puissance 12**

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel

Algorithme 2

réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1 * tmp1 * tmp1 * tmp1 * tmp1
* tmp1 * tmp1 * tmp1 * tmp1 * tmp1 * tmp1

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel

Algorithme 3

réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel

Entrée: 9.5 et 3000

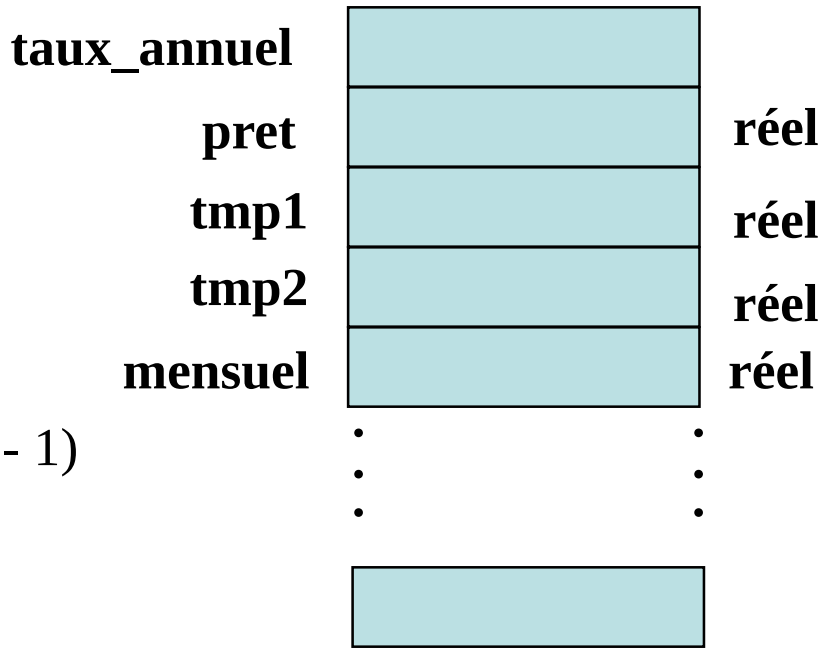
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



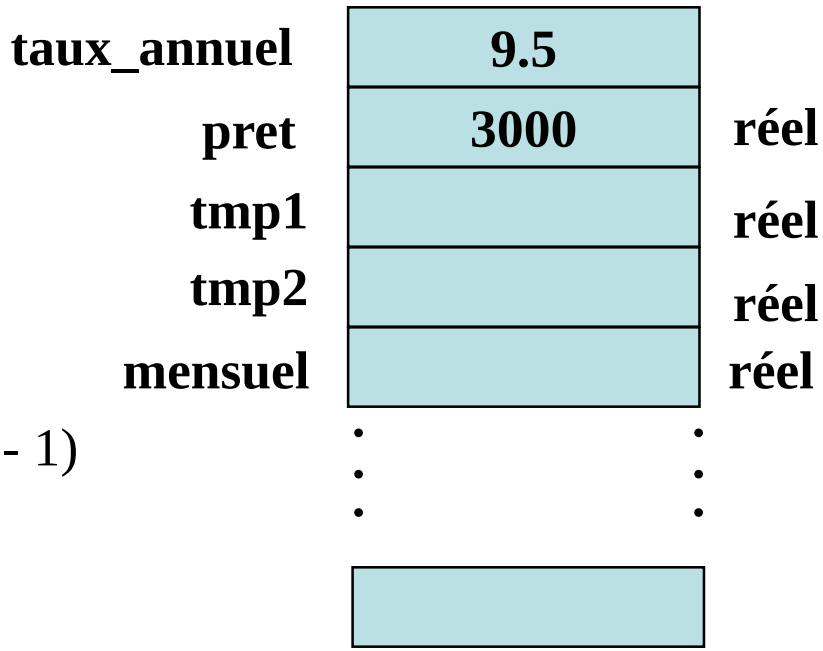
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



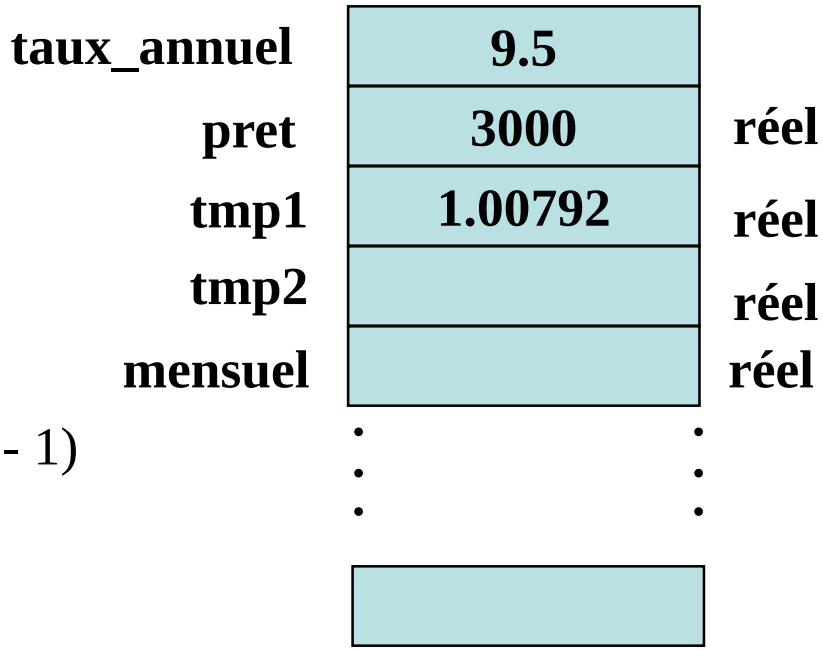
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



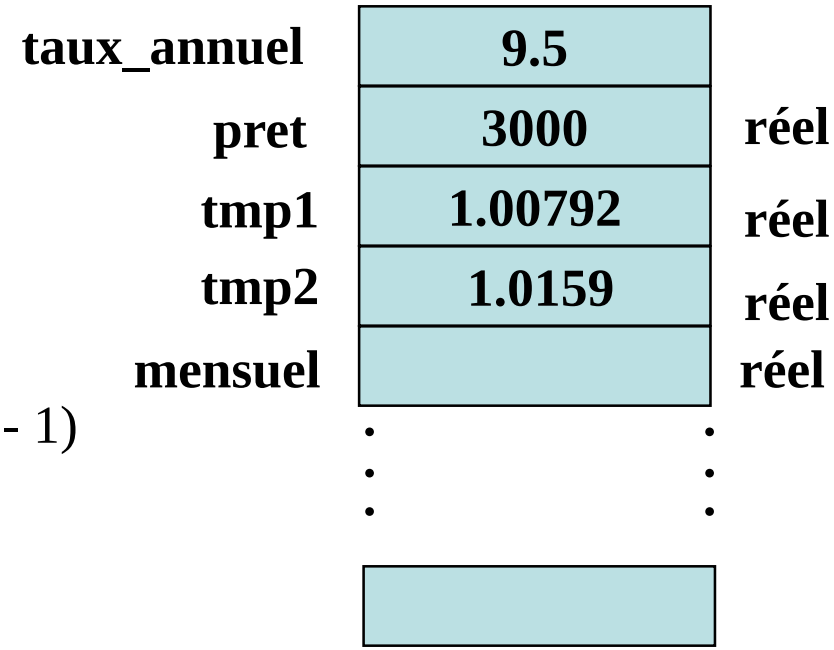
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1

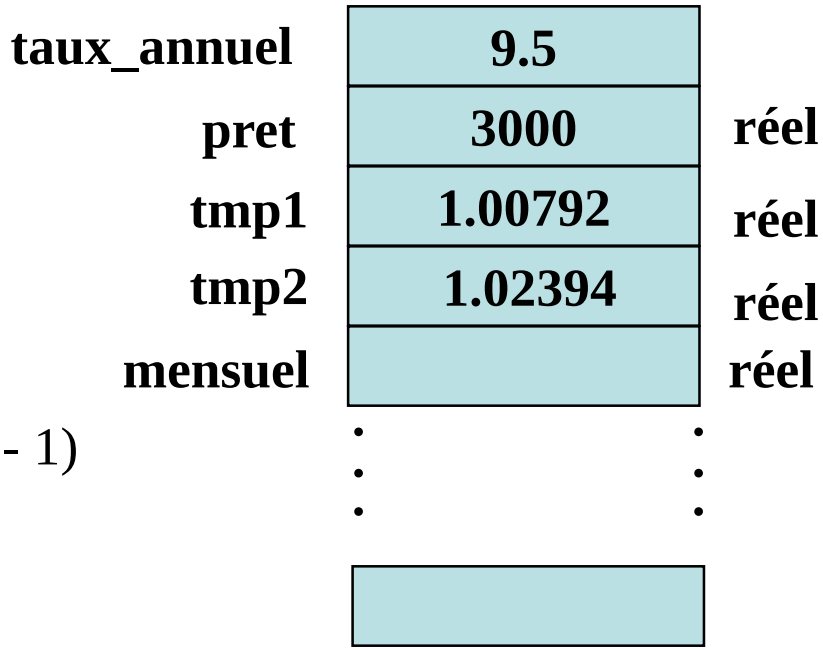
tmp2 ← tmp2 * tmp1

tmp2 ← tmp2 * tmp2

tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)

écrire mensuel



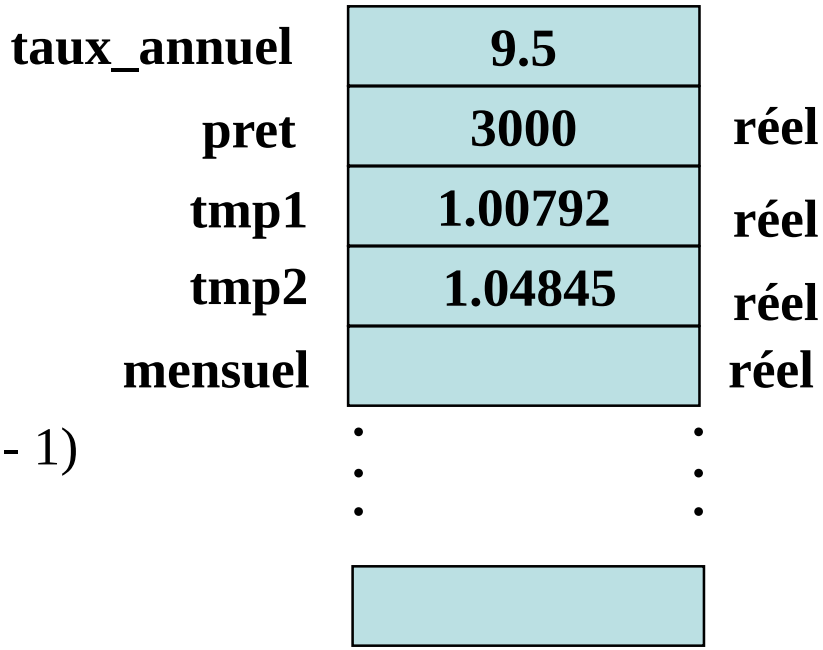
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



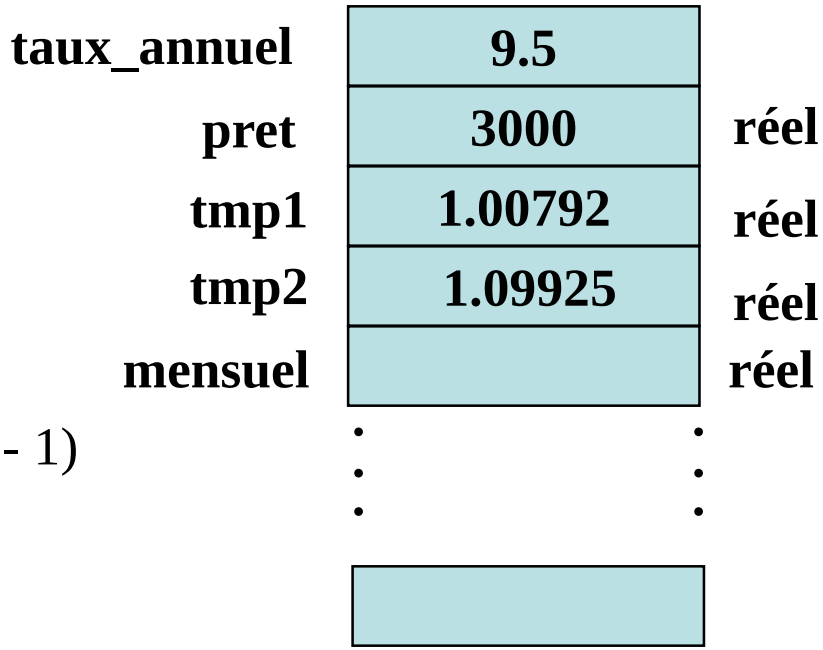
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



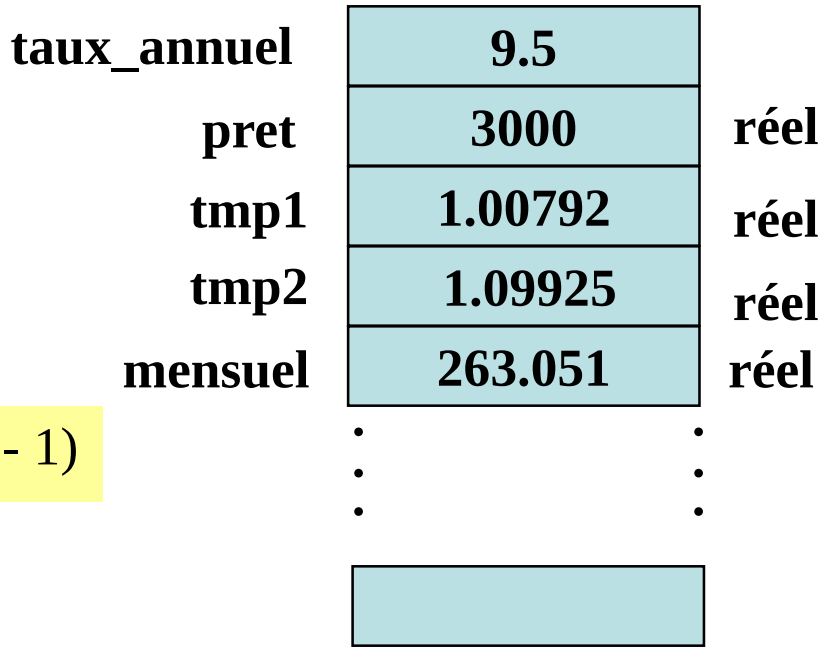
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



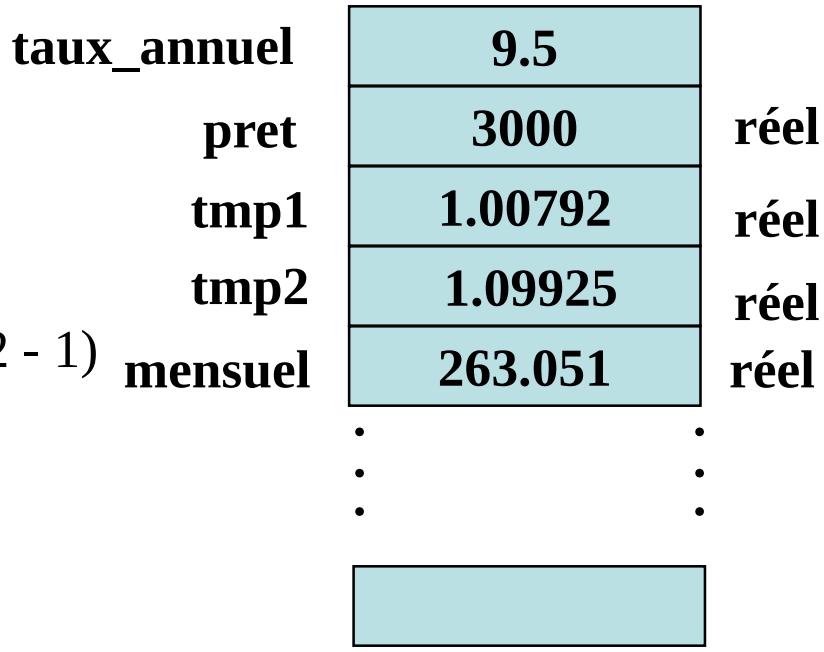
réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel



Sortie: 263.051

Algorithme 3

réel taux_annuel, pret, tmp1, tmp2, mensuel

lire taux_annuel, pret

tmp1 ← 1 + taux_annuel / 1200

tmp2 ← tmp1 * tmp1
tmp2 ← tmp2 * tmp1
tmp2 ← tmp2 * tmp2
tmp2 ← tmp2 * tmp2

mensuel ← pret * (tmp1 - 1) * tmp2 / (tmp2 - 1)
écrire mensuel

Programme en C

```
#include  <stdio.h>
main(){
double taux_annuel, pret, tmp1, tmp2, mensuel;

/* A completer */
tmp1 = 1 + taux_annuel / 1200;

tmp2 = tmp1 * tmp1;
tmp2 = tmp2 * tmp1;
tmp2 = tmp2 * tmp2;
tmp2 = tmp2 * tmp2;

mensuel =  pret * (tmp1 - 1) * tmp2 / (tmp2 - 1);
/* A completer */
return (0);
}
```

Échanger deux valeurs

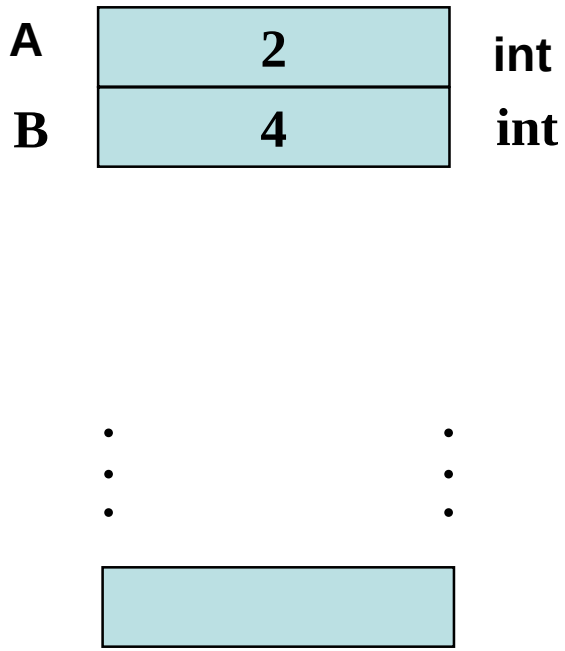
- Soient deux valeurs, contenues dans A et B. Écrire un programme en C qui consiste à échanger les valeurs des ceux variables.

Exemple: en entrée, nous avons la valeur de A = 2 et celle de B = 4. En sortie, on doit avoir A = 4 et B =2.

Solution 1

- **#include** <iostream.h>
- **main**(){
- **int** A, B;
-
- **/* à completer */**

- A = B;
- B = A;
- **/* à completer */**
- **return** (0);
- }



Exécution

- Écrivez votre programme dans un fichier (à l'aide de **emacs**)
- Le nom du fichier doit avoir l'extension **.c** (disons **prog.c**)
- Compilez votre programme à l'aide de la commande

  g++ prog.c
- Vous obtenez un fichier du nom de **a.out**.
- Ce dernier fichier est la traduction de votre programme en langage machine.

  Pour l'exécuter il suffit d'écrire **a.out** suivit d'un retour de chariot.