Constante / Variables

- Constante
  - Ne change jamais de valeur pendant l'exécution d'un programme
  - Constante non typée
    - Macrodéfinition

```
#define NB 5
#define PI 3.14
```

⇨ pas d'allocation en mémoire

⇨ le pré-processeur remplace dans la suite du programme toute référence aux macros définies par leur définition

⇨ le pré-processeur se contente de remplacer les références à la macro par la valeur de celle-ci sans opérer de contrôle sur l 'utilisation de cette valeur

Constante / Variables

- Constante non typée : illustration

TestConst.c

```
/*
 ============================================================
 Name        : TestConst.c
 ============================================================
 */

#include <stdio.h>
#include <stdlib.h>

#define NB_VIES_INITIALES 5

int main(void) {
    int i;
    printf("Nombre de vies initiales: %d\n", NB_VIES_INITIALES);
    for(i=NB_VIES_INITIALES;i>0;i--)
        printf("Il vous reste %d vie(s)\n", i);
    return EXIT_SUCCESS;
}
```

Properties | Problems | Tasks | Console

<terminated> TestConst.exe [C/C++ Application] C:\User

```
Nombre de vies initiales: 5
Il vous reste 5 vie(s)
Il vous reste 4 vie(s)
Il vous reste 3 vie(s)
Il vous reste 2 vie(s)
Il vous reste 1 vie(s)
```

8

Properties | Problems | Tasks | Console

<terminated> TestConst.exe [C/C++ Application] C:\User

```
Nombre de vies initiales: 8
Il vous reste 8 vie(s)
Il vous reste 7 vie(s)
Il vous reste 6 vie(s)
Il vous reste 5 vie(s)
Il vous reste 4 vie(s)
Il vous reste 3 vie(s)
Il vous reste 2 vie(s)
Il vous reste 1 vie(s)
```

Constante / Variables

- Constante typée
  - mot clé const ⇨ const Type NOM_CONSTANTE = ValeurConstante ;

```
const   int     NOMBRE_DE_VIES_INITIALES = 5;
const   float   PI = 3.14;
```

  - Allocation en mémoire du nombre d'octets correspondant au type utilisé pour la déclaration

Constante / Variables

- Constante typée : illustration

TestConst.c

```
/*
 ============================================================
 Name        : TestConstTypee.c
 ============================================================
 */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    const int NB_VIES_INITIALES=5;
    int i;
    printf("Nombre de vies initiales: %d\n", NB_VIES_INITIALES);
    for(i=NB_VIES_INITIALES;i>0;i--)
        printf("Il vous reste %d vie(s)\n", i);
    return EXIT_SUCCESS;
}
```

Properties | Problems | Tasks | Console

<terminated> TestConst.exe [C/C++ Application] C:\User

```
Nombre de vies initiales: 5
Il vous reste 5 vie(s)
Il vous reste 4 vie(s)
Il vous reste 3 vie(s)
Il vous reste 2 vie(s)
Il vous reste 1 vie(s)
```

8

Properties | Problems | Tasks | Console

<terminated> TestConst.exe [C/C++ Application] C:\User

```
Nombre de vies initiales: 8
Il vous reste 8 vie(s)
Il vous reste 7 vie(s)
Il vous reste 6 vie(s)
Il vous reste 5 vie(s)
Il vous reste 4 vie(s)
Il vous reste 3 vie(s)
Il vous reste 2 vie(s)
Il vous reste 1 vie(s)
```

Constante / Variables

- Constante typée : illustration

TestConst.c

```
/*
 ============================================================
 Name        : TestConstTypee.c
 ============================================================
 */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    const float PI=3.14;

    printf("Surface d'un disque de 4 cm de rayon : %.2f\n", PI*4*4);
}
```

Properties | Problems | Tasks | Console

```
<terminated> TestConst.exe [C/C++ Application] C:\Users\fl
Surface d'un disque de 4 cm de rayon : 50.24
```

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    const float PI=3.14;

    printf("Surface d'un disque de 4 cm de rayon : %d\n", PI*4*4);

    return EXIT_SUCCESS;
}
```

format '%d' expects type 'int', but argument 2 has type 'double'

Constante / Variables

- Variables
  - Peut changer de valeur pendant l'exécution du programme
  - Définition de variable
    - Déclaration + réservation de l'espace mémoire
    - liée à
      - La définition du domaine de valeurs et des opérations autorisées

Constante / Variables

- Variables
  - Déclaration de la variable

  ⇔ association nom de la variable et du type
    - Type de la variable
      - char, int, short, float, double …
    - Classe mémoire
      - Globale/locale

      ⇔ Lieu de définition/déclaration
        - En dehors d'une fonction
        - Dans une fonction
      - Qualificatifs
        - register, static, extern
      - Association d'une durée de vie

Constante / Variables

- Variables
  - Initialisation de la variable

  ⇔ affectation d'une valeur initiale
    - Au moment de la définition
    - Dans une instruction séparée

Constante / Variables

- Où et quand définir/déclarer les variables ?

  - **AVANT de les utiliser**
  - En dehors du corps des fonctions

    ⇒ **variables globales**
    ⇒ **visibles dans toutes les fonctions** du fichier où elles sont déclarées
    ⇒ variable est **allouée en début de programme** et est **détruite à la fin du programme**
    ⇒ automatiquement **initialisée à 0**

  - Dans le corps d'une fonction ou au début d' un bloc

    ⇒ **variables locales** ou variables automatiques
    ⇒ **visibles uniquement dans le bloc ou la fonction** où elles sont déclarées et **dans les sous-blocs de celui-ci ou de celle-ci**
    ⇒ variable est **allouée à chaque entrée** dans le bloc et elle est **détruite à la sortie du bloc**
    ⇒ perte de sa valeur à la sortie du bloc
    ⇒ Pas d'initialisation automatique

      valeur initiale doit être **explicitement définie par le programmeur**

☝ **Une variable locale qui a le même nom qu'une variable globale masque la variable globale dans la zone de visibilité de la variable locale**

Constante / Variables

- Domaine de visibilité d'une variable

```c
/*
 ============================================
 Name        : TestVariable.c
 ============================================
 */
#include <stdio.h>
#include <stdlib.h>

int nb;


int main(void) {
    printf("Nb avant affectation: %d\n", nb);
    nb=167;
    printf("Nb après affectation: %d\n", nb);
    return EXIT_SUCCESS;
}
```

Variable Globale

```c
/*
 ============================================
 Name        : TestVariable.c
 ============================================
 */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int i;
    for(i=0;i<3;i++){
        float rayon=5.1 +i;
        printf("rayon  %.2f\n", rayon);
    }
    printf("i: %d",i);
    printf("rayon  %.2f\n", rayon);
    return EXIT_SUCCESS;
}
```

Variable Locale
Visible dans la fonction main

Variable Locale
Visible dans le bloc for

```
Multiple markers at this line
  - each undeclared identifier is reported only once for each function it appears in
  - 'rayon' undeclared (first use in this function)
  - Symbol 'rayon' could not be resolved
```

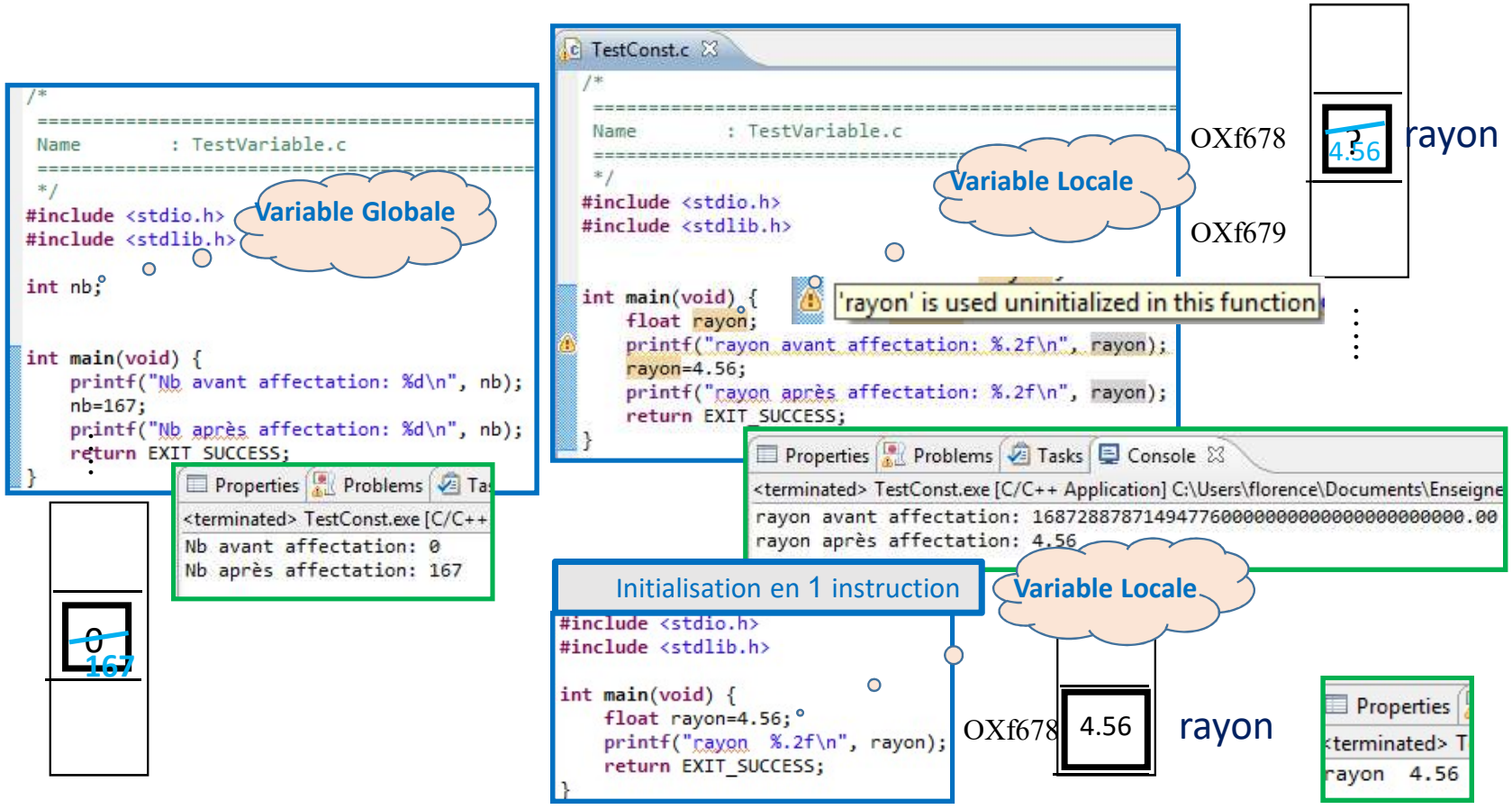
```c
int main(void) {
    int i;
    for(i=0;i<3;i++){
        float rayon=5.1 +i;
        printf("rayon  %.2f\n", rayon);
    }
    printf("i: %d",i);
    return EXIT_SUCCESS;
}
```

Properties

```
<terminated> T
rayon  5.10
rayon  6.10
rayon  7.10
i: 3
```

Constante / Variables

- Définition d'une variable / Initialisation



Constante / Variables

- Classes de variables
  - Mots clés ajoutés devant la définition/déclaration de la variable
    - **extern**
      - Mot clé utilisé pour des variables globales déclarées dans d'autres fichiers
    - **register**
      - variables de type registre sont stockées dans un registre
        ⇒ mémoire à accès plus rapide que la RAM
        ⇒ accélération des traitements mais on ne peut pas accéder à l 'adresse mémoire de la variable
    - **static**
      - variable globale
        ⇒ variable sera invisible dans les autres fichiers
      - variable locale à une fonction
        ⇒ la variable (automatiquement initialisée à 0) n'est pas détruite à la fin de l 'appel de la fonction
        ⇒ elle conserve sa valeur entre 2 appels de fonctions
        ⇒ Variable rémanente

Constante / Variables

- Classes de variables

```c
#include <stdio.h>
#include <stdlib.h>
int somme(int a){
    int s=0;
    s=s+a;
    return s;
}
int main(){
    int i,x;

    for(i=0;i<4;i++){
        printf("Donnez la valeur de x\n");
        scanf("%d",&x);
        printf("La somme des nombres entrés est %d\n", somme(x));
    }
    return EXIT_SUCCESS;
}
```

```c
int somme(int a){
 static int s ;
  s=s+a ;
  return(s) ;
}
```

Exemple d'exécution

i	valeur rentrée	valeur affichée
0	5	5
1	4	4
2	10	10
3	3	3

Exemple d'exécution

i	valeur rentrée	valeur affichée
0	5	5
1	4	9
2	10	19
3	3	22

Université de Paris

COURS 2

Programmation impérative

Langage C – Eléments de base

- les opérateurs
- Les structures de contrôle

SOMMAIRE

Opérateurs

- Opérateurs à 1, 2 ou 3 opérandes : unaires, binaires, ternaires
  - Opérateurs unaires non logiques et non bas niveau

Opérateur	Utilisation	

(type)	cast ou changement temporaire de type
	conversion explicite de type
		int x=45;
		float res;
		res=(float) x;

&	opérateur de déréférencement ou d'adresse retourne l'adresse en mémoire d'une variable	&x;
*	opérateur d'indirection ou de déréférencement sur une adresse permet d'accéder au contenu d'une variable à partir d'une adresse	*(&x);
sizeof	opérateur donnant la taille en octets d'un type	sizeof(int)
-	moins unaire, inversion de signe	int y=-3; x=-y;
++	incrémentation	x++;
--	décrémentation	x--;

Opérateurs

- Opérateurs arithmétiques et d ’affectation binaires

Opérateur	Utilisation
+	Addition arithmétique
-	Soustraction arithmétique
*	Multiplication arithmétique
/	Division entière ou réelle (dépend du type des opérandes)
%	Reste de la division entière 10%8 => 2
=	Opérateur d’affectation

☞ * et / sont prioritaires

Opérateurs

- Opérateurs arithmétiques et Type des opérandes
  - Si les opérandes ne sont pas de même type

⇔le compilateur agrandit en général l'opérande le plus étroit ⇔ conversion implicite

```
int     x=2;
float   res, y=4.5;
res = x+y;
```

⇔ int + float ⇔ le type int est étendu temporairement au type float ⇔ float + float = float

⇔ l 'affectation d 'un type étroit à un type plus large n'entraîne pas de problème alors que l'inverse peut entraîner une perte d'information

```
x=2, y= 6;
float   res;

res = x+y;    //8.0000
```

int + int = int

type du résultat (int)étendu au type float

6 + 2 = 8 **8.0000 <= 8**

```
float   x = 4.5,  y=5.6;
int     res;

res = x + y;  // 10  au lieu de 10.1
```

float + float = float 4.5+5.6=10.1

affectation d'un type large (float) à un type plus étroit au type **int**

⇔4.5+5.6=10.1 perte des chiffres après virgule

10 <= 10.100000

Opérateurs

- Type des opérandes et opérateur /

Type opérandes	Division entière ou réelle ?	int x=2, y= 6; float res;
int / int	**division entière**	res = x / y; ⇔2/6=0 résultat étendu au type float res 0.0000 x 2 y 6
float / int int / float float / float	**division réelle**	res = (float) x / (float) y; ou res = (float) x / y; res 0.33333 x 2 y 6 ou res= x / (float) y

Opérateurs

- Combinaison Opérateurs d'incrémentation/décrémentation et d'affectation

Position de l'opérateur ++/variable à incrémenter (même chose pour --)	**Quelle opération en 1er ?**	**int v1, var =2;** v1 ? var 2

v1 = var++;
⇔
v1= var;
var++;
	var est incrémentée après l 'affectation	v1 2 var 2 puis v1 2 var 3

v1 = ++var;
⇔
var++;
v1= var;
	var est incrémentée avant l 'affectation	v1 ? var 3 puis v1 3 var 3

Opérateurs

- Combinaison Opérateurs d'arithmétique et d'affectation

Opérateurs combinés	Instructions équivalentes	int var =10; var 10

+=	var += 4;
	⇔
	var = var + 4;
		var 14

-=	var -= 4;
	⇔
	var = var - 4;
		var 6

*=	var *= 4;
	⇔
	var = var * 4;
		var 40

/=	var /= 4;
	⇔
	var = var / 4; //division entière
		var 2

%=	var %= 3;
	⇔
	var = var%3; //reste division
	//entière
		var 1