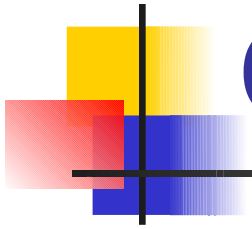
Cours 3

Opérateurs relationnels

Il existe en C six opérateurs relationnels résumés dans le tableau ci-contre :

Opérateur	**Signification**
<	Inférieur strict
<=	Inférieur ou égal
>	Supérieur strict
>=	Supérieur ou égal
==	Égal
!=	Différent

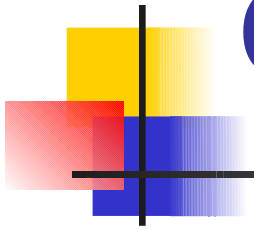


Opérateurs relationnels

Attention : ne pas confondre

- le signe = de l'affectation
- et le signe == de l'égalité

Cela est source d'erreurs difficiles à déceler



Opérateurs relationnels

- En C il n'y a pas de type « booléen » {Vrai, Faux}
- Le résultat d'une comparaison est un entier qui vaut :
  - 0 si le résultat est faux
  - 1 si le résultat est vrai

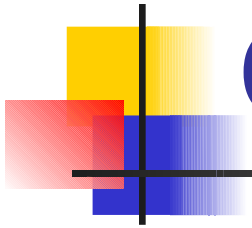
Opérateurs relationnels

- Exemple : soit les déclarations suivantes

```
int n = 0, i = 1, k = -37;
float x = 0.5;
char c1 = 'a', c2 = 'b';
```

- Donnez les valeurs des expressions booléennes :

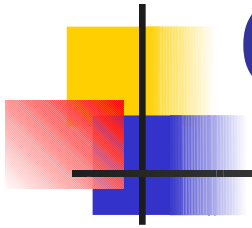
Expression	Valeur
`n == k`	
`n >= k`	
`x < i`	
`c2 > c1`	
`i != k`	
`n <= (i+k)`	
`(n >= i) == (i != n)`	



Opérateurs logiques

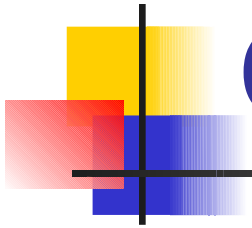
Il existe en C trois opérateurs logiques résumés dans le tableau ci-contre :

Opérateur	**Signification**
!	Non
&&	Et
||	Ou (inclusif)



Opérateurs logiques

- Les opérandes sont des valeurs numériques
- 0 est considéré comme « faux », toute valeur non nulle comme « vrai »



Opérateurs logiques

Table de vérité de **!** (Non)

op	**! op**
0	1
non nul	0

Opérateurs logiques

Table de vérité de `&&` (Et)

`&&`	0	non nul
0	0	0
non nul	0	1

Attention : `&&` n'évalue son deuxième opérande que si nécessaire

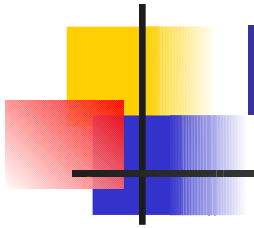
Opérateurs logiques

Table de vérité de || (Ou)

||	0	non nul
0	0	1
non nul	1	1

Attention : || n'évalue son deuxième opérande que si nécessaire

Opérateurs relationnels et logiques



- Définition : une année est bissextile si elle est divisible par quatre à l'exception des années multiples de 100 non divisibles par 400

- Problème : écrire un programme qui imprime 1 si une année donnée est bissextile, 0 sinon

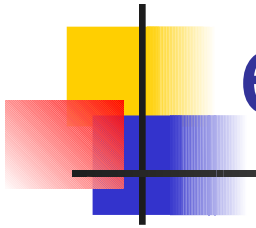
Année bissextile

- D'après la définition l'année n est bissextile si et seulement si :
  n est divisible par 4 ***et***
  non (n est divisible par 100 ***et*** n n'est pas divisible par 400)

- On en déduit le programme suivant :

```
#include <stdio.h>

 int main() {

    int annee; // l'annee a tester
    printf("Donnez l\'annee : ");
    scanf("%d", &annee); // lecture de l'annee au clavier



    return 0;
}
```

Une structure de donnée efficace : le tableau



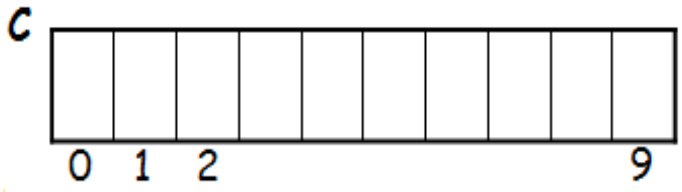
- On prélève 10 condensateurs d'un lot et l'on mesure leur capacité
- On obtient une séquence de résultats C1, C2, …, C10
- Pour mémoriser ces 10 nombres on peut :
  - Définir 10 variables c1, c2, …, c10
  - Définir un tableau nommé c de taille 10 où chaque valeur est repérée par son indice

Une structure de donnée efficace : le tableau

- Pour définir un tel tableau on déclare :

```
float c[10];
```

- Un tableau peut-être vu comme un ensemble de cases contiguës :



- **<u>Attention</u>** : l'indice de la première case est **<u>0</u>**

Une structure de donnée efficace : le tableau

- On accède aux éléments d'un tableau en utilisant le nom du tableau , l'opérateur `[]` et un indice entier compris entre 0 et n-1, où n est la taille du tableau (nombre de ses éléments)

- La valeur du 3ème condensateur est par exemple `c[2]`, celle du 10ème `c[9]`

- La norme ANSI précise que tout accès à un élément situé en dehors du tableau (mauvais indice) conduit à un comportement indéfini du programme !!

Une structure de donnée efficace : le tableau

- Dans un programme la taille d'un tableau peut être souvent considérée comme une constante
- Néanmoins, en C (*), on ne peut définir la taille d'un tableau en utilisant une constante entière définie par le mot clé **`const`**
- On utilise alors le préprocesseur et la directive **`#define`** qui permet la substitution de symbole :

```
#define N 100
int tableau[N];
```

- Dans le programme **`N`** sera partout remplacé par 100 avant la compilation effective (preprocessing). Pour modifier la taille du tableau, il suffit de modifier 1 ligne !

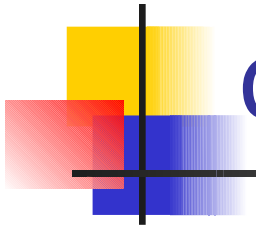
(*) C'est possible en C++

Agréger des données de type différents : la notion de structure

- Supposons que l'on désire travailler avec des points colorés du plan (abscisse, ordonnée, couleur). En C, on peut définir un type **`point`** comme l'agrégat de deux réels (abscisse, ordonnée) et de trois entiers codant la couleur (codage RGB)

```
typedef struct {
    float x; // memorise l'abscisse
    float y; // memorise l'ordonnée
    int r,g,b; // memorise la couleur
} point_colore;
```

Agréger des données de type différents : la notion de structure



- Il faut « lire » la définition précédente de la manière suivante :

- je définis un type (**`typedef`**)
- qui est structuré (**`struct`**)
- et dont les valeurs sont un 5-uple composé de
  - deux réels nommés **`x`** et **`y`**
  - et de trois entiers **`r`**, **`g`**, **`b`** qui définissent la couleur du point
- et ce type s'appellera **`point_colore`**

Agréger des données de type différents : la notion de structure

- On peut alors déclarer des variables de type **`point_colore`**

```
point_colore p; //point((1.5,0),rouge)
```

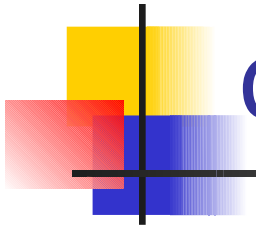
- Pour accéder aux champs de **`p`** on utilise la notation «.». On peut alors initialiser p en initialisant ses champs :

```
p.x = 1.5; p.y = 0;
p.r = 255; p.g = p.b = 0;
```

- On peut aussi initialiser directement p de la manière suivante :

```
p = {1.5, 0, 255, 0, 0};
```

Agréger des données de type différents : la notion de structure



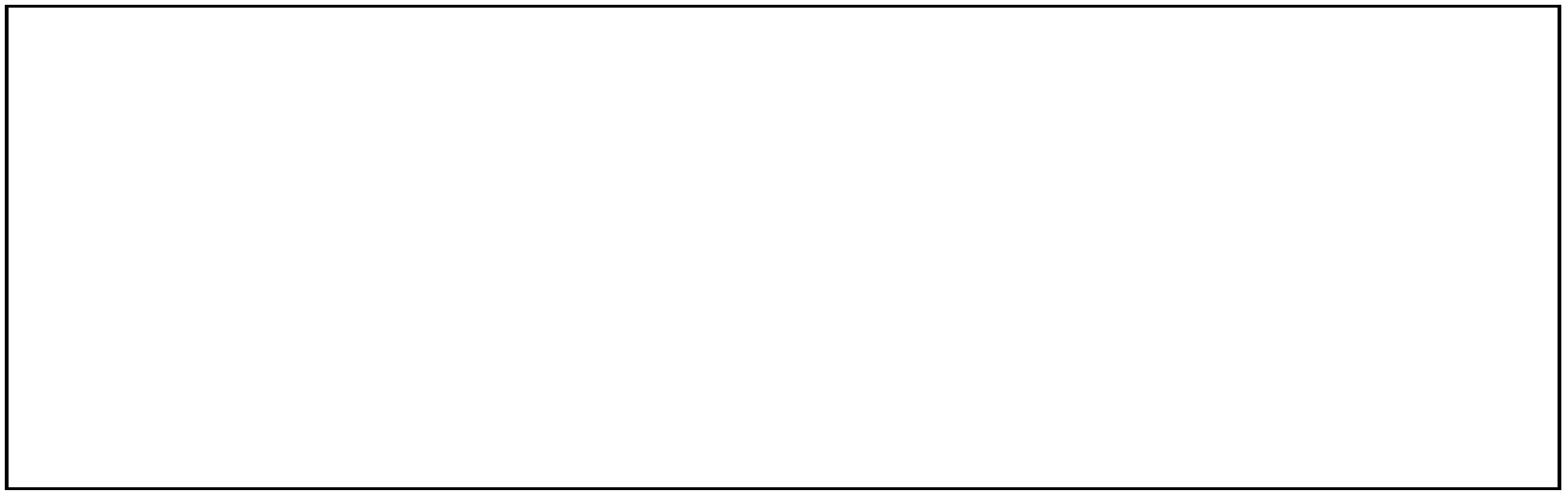
Remarque : finalement C vous propose

- Des types de bases : **`int`**, **`float`**, **`double`**, **`char`**

- Et vous permet de fabriquer vos propres types en utilisant la notion de structure

Un autre exemple de type structuré : les nombres complexes

- Définissons un type structuré **`complexe`** permettant de mémoriser des nombres complexes sous forme algébrique :



Un autre exemple de type structuré : les nombres complexes

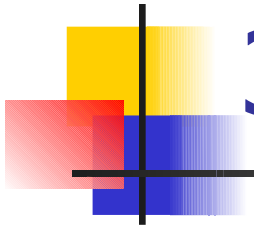
- Exercice : on veut mémoriser dans deux variables **`z1`** et **`z2`** les complexes 1 + i et 2 – 5i, puis dans une variable **`z`** la somme **`z1+z2`**. Écrivez le code correspondant :

Des tableaux de structures

- Un tableau peut être composé de valeurs d'un type structuré
- Par exemple on peut déclarer un tableau **`nuages`** de 100 points colorés :

```
point_colore nuages [N];
```

- Exercice : affectez au 5ème point coloré de ce tableau les coordonnées (1.0, -1.0) et la couleur bleu :



3.2 Les structures de contrôle

- Dans un programme les instructions sont exécutées séquentiellement

- Pour traiter l'information de manière effective et « intelligente » on aura également besoin :

  - D'effectuer des choix (conditionnelle)
  - D'effectuer des itérations

- Les instructions du langage qui permettent cela s'appelle structures de contrôle car elles contrôlent l'enchaînement des instructions

Les instructions en C

En C on considère des instructions :

- De déclaration, qui fournissent des informations au compilateur.

  Exemple : `int n; // declaration d'une variable`
  `// nommee n de type int`

- Exécutables , qui implique un traitement.

  Exemple : `n = n + 1; // incrementation de 1 de la`
  `// variable n`

Les instructions en C

- Instruction expression : [expression]**;**
  - Exemples :
    - `int n = 1;`
    - `printf(« bonjour\n »);`
    - `y = (x*x) + x + 1;`

- Instruction composée ou bloc :

{

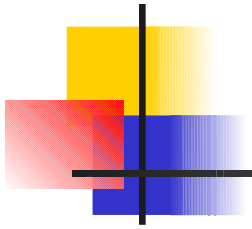
[déclarations]

[0, 1 ou plusieurs instructions exécutables en séquence]

}

La conditionnelle

- Il s'agit en fonction d'une réponse utilisateur, du résultat d'un calcul, etc. :
  - D'effectuer un traitement
  - D'effectuer un traitement ou un autre

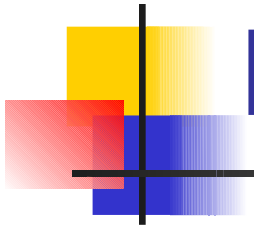


La conditionnelle

- Les deux formes de l'instruction **`if`**

```
if (expression)
  instruction_1
```

```
if (expression)
  instruction_1
else
  instruction_2
```



La conditionnelle

Remarque : une instruction dans une conditionnelle peut être instruction expression ou instruction composée

- Règle de bonne programmation (*) : toujours considérer une instruction dans une conditionnelle comme composée, en clair toujours ouvrir et fermer une accolade

La conditionnelle

- On peut également enchaîner les conditionnelles avec la construction **`else if`**

- Exemple : étudiez le signe d'un nombre

```
if (nombre < 0){
   signe = '-';
}
else if (nombre > 0) {
   signe = '+';
}
else {
   signe = ' ';
}
```

La conditionnelle

- Quand la valeur à affecter à une variable dépend d'une condition on peut utiliser l'opérateur conditionnel

- Exemple : considérons l'instruction :

```
if (a>b) {
   max = a;
}
else {
   max = b;
}
```

Elle peut s'écrire plus simplement :

```
max = (a>b) ? a : b;
```

Exemple : système d'alerte anticollision pour les avions

- Problème : éviter que des avions se percutent en vol

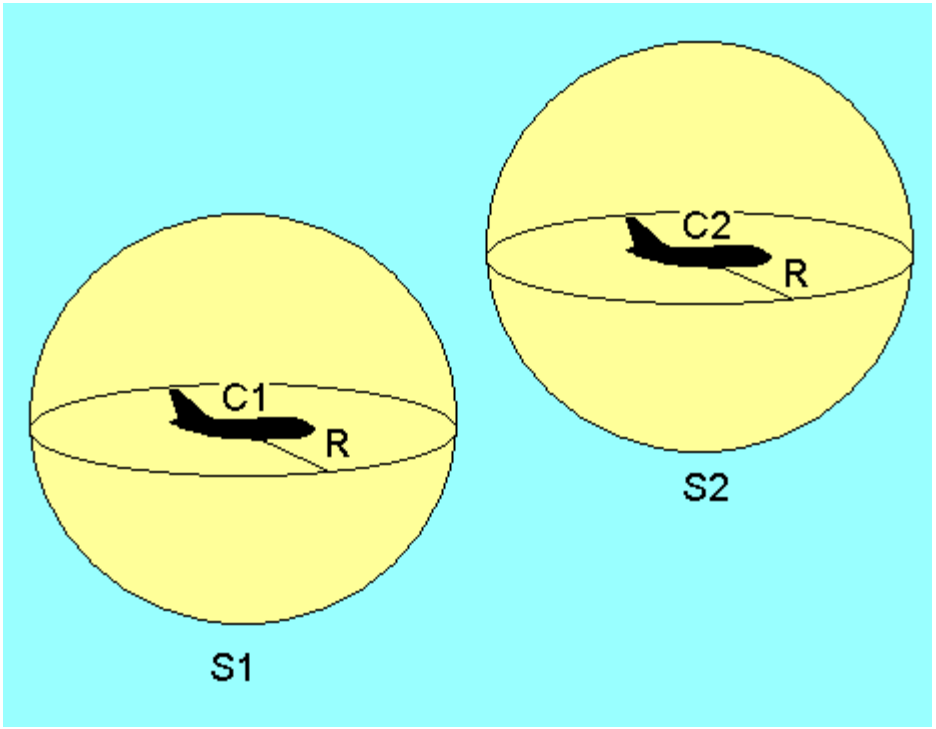
- De tels systèmes existent (ACAS : Airborne Collision Avoidance System, TACAS : Traffic Collision Avoidance System basés sur les transpondeurs (http://perso.wanadoo.fr/controleaerien/xpdr.html) des avions)

- Étudions une version très simplifiée qui serait basée sur les systèmes GPS + altimètre

Exemple : système d'alerte anticollision pour les avions

- Un GPS plus un altimètre permet de donner une position spatiale d'un avion (x, y, z)
- Si l'on considère une sphère de protection de rayon R suffisant centrée sur l'avion le système d'alerte revient en temps réel à :
  - Tester si les sphères de deux avions s'intersectent ou pas
  - Prévenir les pilotes si leurs sphères de sécurité s'intersectent

Exemple : système d'alerte anticollision pour les avions



Exemple : système d'alerte anticollision pour les avions

- Soient S_1 et S_2 les sphères de rayons R et de centre $C_1(x_1,y_1,z_1)$, $C_2(x_2,y_2,z_2)$ les sphères s'intersectent si et seulement si :

$$d(C_1, C_2) < 2R$$

- Avec : $$d(C_1, C_2) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$$

Exemple : système d'alerte anticollision pour les avions

```
const float R = 1.0f; // rayon de securite en km
float d;
float x1,y1,z1,x2,y2,z2;

d = sqrt(((x2-x1)*(x2-x1)) + ((y2-y1)*(y2-y1)) +
         ((z2-z1)*(z2-z1)));

if (d >= ((2 * R) + .1)) {
    printf("OK\n");
}
else if (fabs(d - (2*R)) < .1) { // on est aux limites
    printf("Warning … \n");
}
else {
    printf("Too near (do something !!?) ! \n");
}
```

Exemple : résolution de l'équation du second degré dans $\mathbb{R}$

- Il s'agit d'écrire un programme C qui résout l'équation du second degré dans $\mathbb{R}$

- On calcule le discriminant et selon son signe on effectue différents traitements. On est bien dans le cadre d'une programmation avec conditionnelle

Exemple : résolution de l'équation du second degré dans $\mathbb{R}$

- Rappel de la résolution mathématique :

$$ax^2 + bx + c = 0$$

On calcule $\Delta = b^2 - 4ac$

- Si $\Delta < 0$ pas de solution
- Si $\Delta = 0$ une solution double $x = -b / 2a$
- Si $\Delta > 0$ deux solutions
  - $x_1 = (-b + \sqrt{\Delta}) / 2a$
  - $x_2 = (-b - \sqrt{\Delta}) / 2a$

Exemple : résolution de l'équation du second degré dans $\mathbb{R}$

- Implantation de la résolution :
  - On a besoin du calcul de la racine carrée, donc on inclut **`math.h`**
  - On déclare trois variables réelles **`a`**, **`b`**, **`c`** mémorisant les coefficients du trinôme
  - On déclare également des variables pour la mémorisation du discriminant, de sa racine (éventuelle) et des (éventuelles) deux solutions

Le programme : déclarations, lecture des données

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

int main() {
    double a,b,c; // les coefficients de l'équation
    double delta, rac_delta; // le discriminant et sa racine
    double x1,x2; // les deux solutions

    // lecture des coefficients
    printf ("Donnez les trois coefficients : ");
    scanf("%lf%lf%lf", &a,&b,&c);
```

Le programme : résolution de l'équation

```
if (a == 0) { // cas ou a est nul
    printf("Pas une equation du second degre !\n");
}
else { // calcul du discriminant
    delta = (b*b) - (4*a*c);









  }
  return 0;
}
```