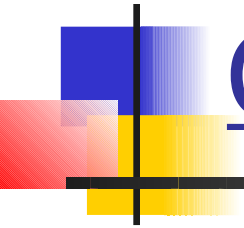
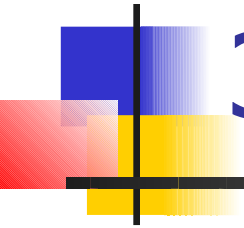
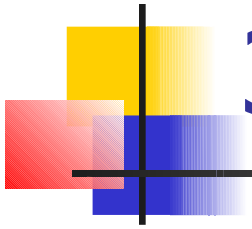


Cours 2



3. Les bases du langage C

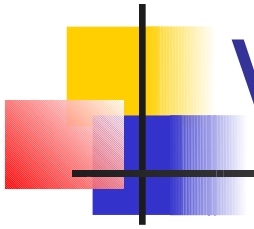


3.1 Les données

- Variables, constantes
- Types de base et opérations
- Les entrées-sorties
- Opérateurs relationnels et logiques
- Tableaux, Structures

Variables, constantes

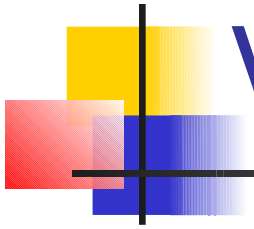
Un programme informatique traite des données et fournit des résultats



Variables, constantes

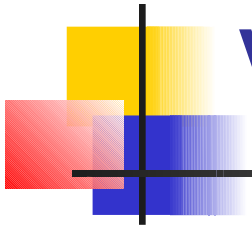
- Les données (et également les résultats) sont stockés dans des variables

- Quand le contenu d'une variable ne doit pas changer durant l'exécution du programme, on parle de constante



Variables, constantes

- Une variable peut-être vue comme une « case » mémoire
- Elle contient une information
- L'information est accessible via le nom ou identificateur de la variable



Variables, constantes

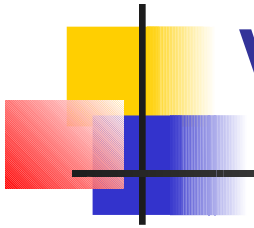
Règles de formation des identificateurs en C :

- Lettres (majuscules et minuscules différenciées), chiffres et _ (« souligné »)
- Premier caractère différent d'un chiffre
- Ne doit pas être un mot clé du langage

Variables, constantes

Les mots clés de langage C (en gras, ceux que vous connaîtrez à l'issue de ce cours)

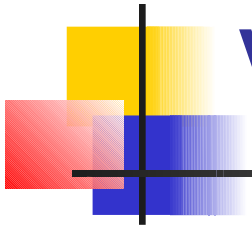
auto	default	**float**	register	**struct**	volatile
break	**do**	**for**	**return**	switch	**while**
case	**double**	goto	short	**typedef**	
char	**else**	**if**	signed	union	
const	enum	**int**	sizeof	unsigned	
continue	extern	long	static	**void**	



Variables, constantes

Validité des identificateurs

Identificateur	Valide	Non valide
`frequence`		
`frequence_2`		
`FREquence`		
`_longueur_onde`		
`%inductance`		
`temps_de_demi_vie`		
`_14_carbone`		
`volatile`		
`volatile_66`		
`14_C`		
`-puissance-moy`		



Variables, constantes

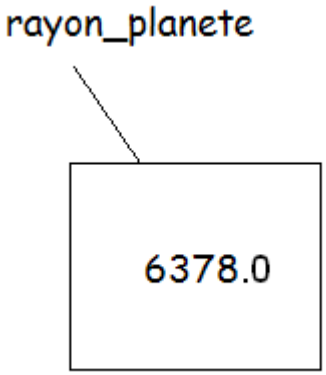
Règle de bonne programmation :

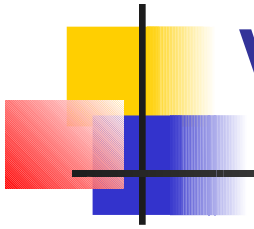
Choisir des identificateurs de variables qui correspondent à l'information qu'elles mémorisent

Variables, constantes

- La variable ci-contre a pour identificateur rayon_planete

- Elle contient le réel 6378.0 (rayon de la Terre)



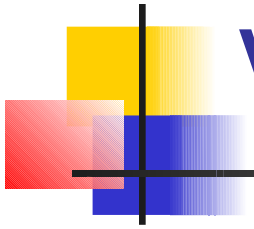


Variables, constantes

- Pour changer (ou initialiser) le contenu d'une variable on utilise l'opération d'affectation :

```
rayon_planete = 6378.0;
```

- Remarque : l'affectation est une instruction C, elle se termine obligatoirement par un « ; »



Variables, constantes

- Attention, le signe « = » de l'affectation n'a pas le sens mathématique usuel. Il induit une notion de temps

- Par exemple, analysons l'affectation suivante où n est une variable de contenu entier : `n = n + 1;`

Autopsie d'une affectation

- Supposons qu'avant exécution de l'instruction d'affectation `n = n + 1`, la variable n contienne la valeur entière 7 :

  n | 7 |

- L'instruction s'exécute en deux temps :
  - Evaluation de l'expression n + 1 en utilisant le contenu actuel de n (7)
  - Remplacement du contenu actuel de n (7) par l'expression évaluée (8)

- La variable n contient maintenant la valeur 8 :

  n | 8 |

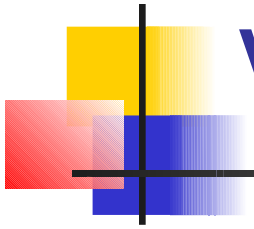
Variables, constantes

- Le programme ci-contre travaille sur des variables contenant des entiers.
- Exécutons le à la main pour donner le contenu de toutes les variables en fin d'exécution

```
int main() {
    int a, b, c, d;
    a = 1;
    b = 2;
    c = d = -5;
    a = a + 3;
    b = a - b + 1;
    b = b + c;
    a = a - 1;
    c = (2*a) - b + c;
    d = (a + b) * c;
    return 0;
}
```

Variables, constantes

Instruction	a	b	c	d
`a = 1;`	1	?	?	?
`b = 2;`	1	2	?	?
`c = d = -5;`	1	2	-5	-5
`a = a + 3;`				
`b = a - b + 1;`				
`b = b + c;`				
`a = a - 1;`				
`c = (2*a) - b + c`				
`d = (a + b) * c;`				

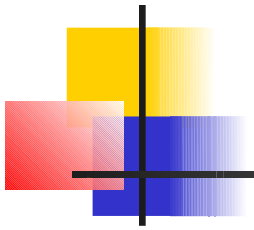


Variables, constantes

- Quand une information a une valeur stable durant l'exécution du programme, on dit que c'est une constante
- En C, on précise qu'une variable est constante en utilisant le mot réservé **`const`**
- Par exemple, π est constant :

```
const float PI = 3.14;
```

- Une « variable » constante doit être initialisée
- Une instruction comme **`PI = PI + 1`** sera rejetée par le compilateur



Types de base

- Une variable (ou une constante) stocke une information dont les valeurs appartiennent à un domaine de valeurs ou *type*
- Par exemple la constante **`PI`** stocke une valeur de type **`float`**
- Le type **`float`** correspond à un sous-ensemble de l'ensemble des nombres décimaux contenus (pour les positifs) au minimum dans l'intervalle $[10^{-37}; 10^{37}]$ ayant au moins 6 chiffres significatifs

Types de base

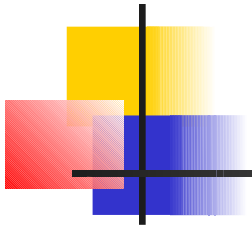
- On ne peut déclarer une variable sans indiquer son type

- La syntaxe est donnée par :

  `type_variable nom_variable;`

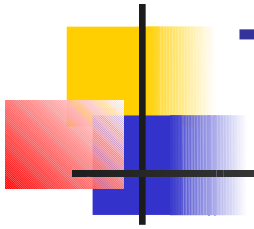
- Exemples :

  - `int n; // la variable n de type entier`
  - `float x; // la variable x de type reel`
  - `int k = 1; /* la variable k de type entier initialisee à 1 */`



Types de base

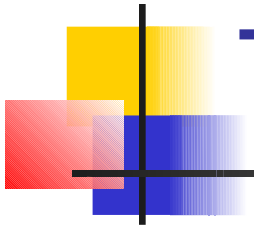
Le langage C possède des types de bases *numériques* et un type de base *caractère*



Types de base numériques

- C possède de nombreux types numériques, nous donnons les trois plus utiles

- Le type **`int`** correspond au minimum au sous intervalle [-32 768, 32 767] de $\mathbb{Z}$



Types de base numériques

- Le type **`float`** (déjà vu)

- Le type **`double`** : identique au type **`float`** mais le nombre minimum de chiffres significatifs est de 10. Il est donc utilisé quand une grande précision est nécessaire

Types de base numériques

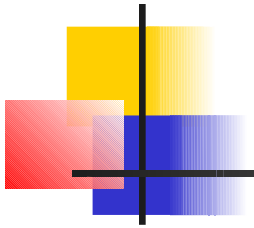
- On écrit les nombres de type **`float`** ou **`double`** en utilisant un point et/ou la notation exponentielle

- Voici des exemples corrects :

  11.77 , -0.056 , .25 , 6. , 11e5 (1 100 000),
  -57.32e-7 (-0.000005732)

- Par défaut toutes les constantes numériques non entières sont crées par le compilateur comme des **`double`** sauf indication explicite par la lettre **`f`**

  (exemple : **`const float PI = 3.14f;`**)

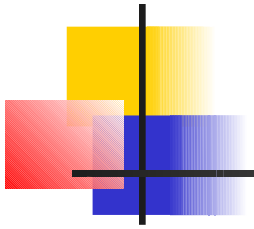


Le type **`char`**

- Le type **`char`** est l'ensemble des caractères (lettres, chiffres, ponctuations, etc.)

- La norme ANSI impose un jeu minimal de caractères :
  - Lettres (non accentuées) de l'alphabet latin, majuscule et minuscules, chiffres décimaux, « espace »
  - Quelques autres caractères (exemple : ! % & , ; : / < > = # [ ] ^ { } ~)
  - Des caractères dits de contrôle (exemple '\n' : retour à la ligne)

- Déclaration d'une variable de type **`char`** initialisée au caractère x : **`char c = 'x';`**

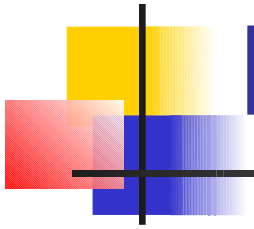


Et les chaînes de caractères ?

- Il n'y a pas de type de base « chaîne de caractères » en C
- On peut néanmoins les manipuler sous forme de tableaux de caractères, mais cela dépasse le cadre de ce cours
- Les seules chaînes de caractères utilisées ici seront les chaînes constantes paramètres de la fonction **`printf`**
- Exemple :

```
printf("%s%f%s\n", "Volume : ", volume, " km3");
```

Opérations sur les types de base numériques

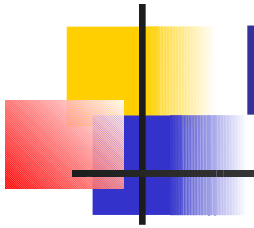


- Les opérations arithmétiques de base (`+`, `-`, `*`, `/`) sont disponibles pour les variables de type numérique

- Il y a un ordre de priorité sur les opérations

- Règle de bonne programmation : parenthéser les expressions algébriques pour lever les ambiguïtés
  Exemple : on écrira `a+(b*c)` plutôt que `a+b*c`

Opérations sur les types de base numériques



- Pour le type **`int`** (entier relatif) on dispose de l'opérateur **`modulo`** « **`%`** » qui rend le reste dans la division euclidienne

- Attention : quand les deux opérandes sont des entiers l'opérateur de division **`/`** est associé à la division euclidienne

- Par exemple à la fin de l'exécution des trois lignes suivantes **`r`** contient 4 et **`d`** contient 3 (et non pas 3.8 !):

```
int a = 19, b = 5, r, d;
r = a % b;
d = a / b;
```

Opérations sur les types de base numériques

- Attention, si l'un des opérandes est négatif le résultat est «compilateur dépendant »

- On peut écrire :

  -9 = (-1 x 5) + (-4) ou -9 = (-2 x 5) + 1

  Dans le premier cas la machine donnera -9 % 5 = -4 et -9 / 5 = -1, dans le second -9 % 5 = 1 et -9 / 5 = -2

- Dans tous les cas on aura toujours :

  a = (b * (a/b)) + (a%b)

Un exemple d'utilisation de l'opérateur %

- La fonction **`rand()`** (cf. TP 2) renvoie un nombre entier aléatoire entre 0 et **`RAND_MAX`** (au moins égal à 32767)

- L'instruction ci-dessous affecte à la variable de type entier **`n`** un nombre aléatoire entre 1 et 10 :

```
int n;
n = (rand() % 10) + 1;
```

Entier aléatoire dans l'intervalle [a;b]

- Écrivez la ligne de code qui permet d'affecter à la variable **`n`** un nombre entier aléatoire entre les entiers relatifs **`a`** et **`b`** (**`a < b`**)

Incrémentation d'une variable numérique

- Pour incrémenter de 1 une variable numérique x on peut écrire :

```
x = x + 1;
```

- C propose un opérateur d'incrément à 1 : « ++ ». On écrira plus simplement :

```
x++;
```

- On dispose également de l'opérateur de décrément à 1 « -- »

Les entrées-sorties : la fonction `printf`

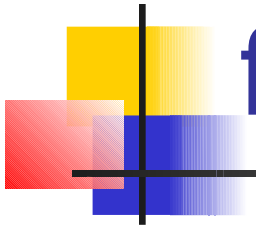
- La fonction **`printf`** permet d'afficher sur la sortie standard (l'écran) le contenu des variables manipulées
- C'est une fonction à arguments variables c'est-à-dire que le nombre de ses arguments et leur type (nombre, caractère) est variable. Cela permet d'afficher plusieurs valeurs en un même appel à **`printf`**
- Le premier argument (ou format) décrit dans une chaîne de caractères la suite des types des valeurs à afficher
- Les arguments suivants, séparés par des virgules, sont les valeurs à afficher. Ils doivent être en accord avec le format spécifié

Les entrées-sorties : la fonction `printf`

- Le tableau ci-contre donne le format et le type de valeur correspondante et éventuellement sa mise en forme
- Dans le cas de l'affichage d'une seule chaîne de caractères constante, le format n'est pas nécessaire

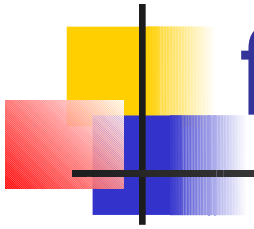
Format	*Type*
`%d`	**`int`**
`%f`	**`float, double`**
`%e`	**`float, double`** (notation exp.)
`%c`	**`char`**
`%s`	chaîne cst

Les entrées-sorties : la fonction `printf`



- Les valeurs numériques affichées peuvent être formatées en agissant sur la syntaxe du format. On peut essentiellement :
  - Agir sur le gabarit de la valeur
  - Agir sur la précision

Les entrées-sorties : la fonction `printf`



- Le gabarit : c'est le nombre minimum de caractères affichés. On l'indique avant le caractère de format

- Exemple : `%7f` indique que le réel doit être affiché sur au moins 7 caractères

Les entrées-sorties : la fonction `printf`

- La précision : c'est le nombre de chiffres après le point décimal. Par défaut, `%f` affiche 6 chiffres après le point décimal. On peut imposer n chiffres après le point décimal en ajoutant .n avant le caractère de format

- Exemple : `%.3f` indique que le réel doit être affiché avec trois chiffres après le point décimal

Les entrées-sorties : la fonction `printf`

- On peut bien sûr agir sur le gabarit et la précision

- Exemple : `%9.3f` indique que le réel doit être affiché sur au moins 9 caractères avec 3 chiffres après le point décimal

- Comment s'affichera à l'écran les variables `x` et `y` de type `double`, de valeur respectives en mémoire 3.21 et 1274.5678 après exécution de l'instruction `printf("%9.3f\n %9.3f\n",x,y);` ?

Les entrées-sorties : la fonction `scanf`

- Le tableau ci-contre donne le format et le type de valeur correspondante

Format	*Type*
`%d`	`int`
`%f`	`float`
`%lf`	`double`
`%c`	`char`

Les entrées-sorties : la fonction **`scanf`**

- Le fonctionnement de **`scanf`** est plus compliqué que celui de **`printf`**
- Même si l'on peut lire plusieurs variables en un seul **`scanf`** on ne le fera pas dans ce cours
- Quand on lit une variable de type caractère, ***`scanf`*** considère les espaces et le retour à la ligne comme un caractère. On prendra donc soin de taper au clavier un caractère unique puis « Entrée », sans espace devant le caractère tapé
- Attention : n'oubliez pas le caractère **`&`** devant le nom de la variable (faute de mémoire sinon !)