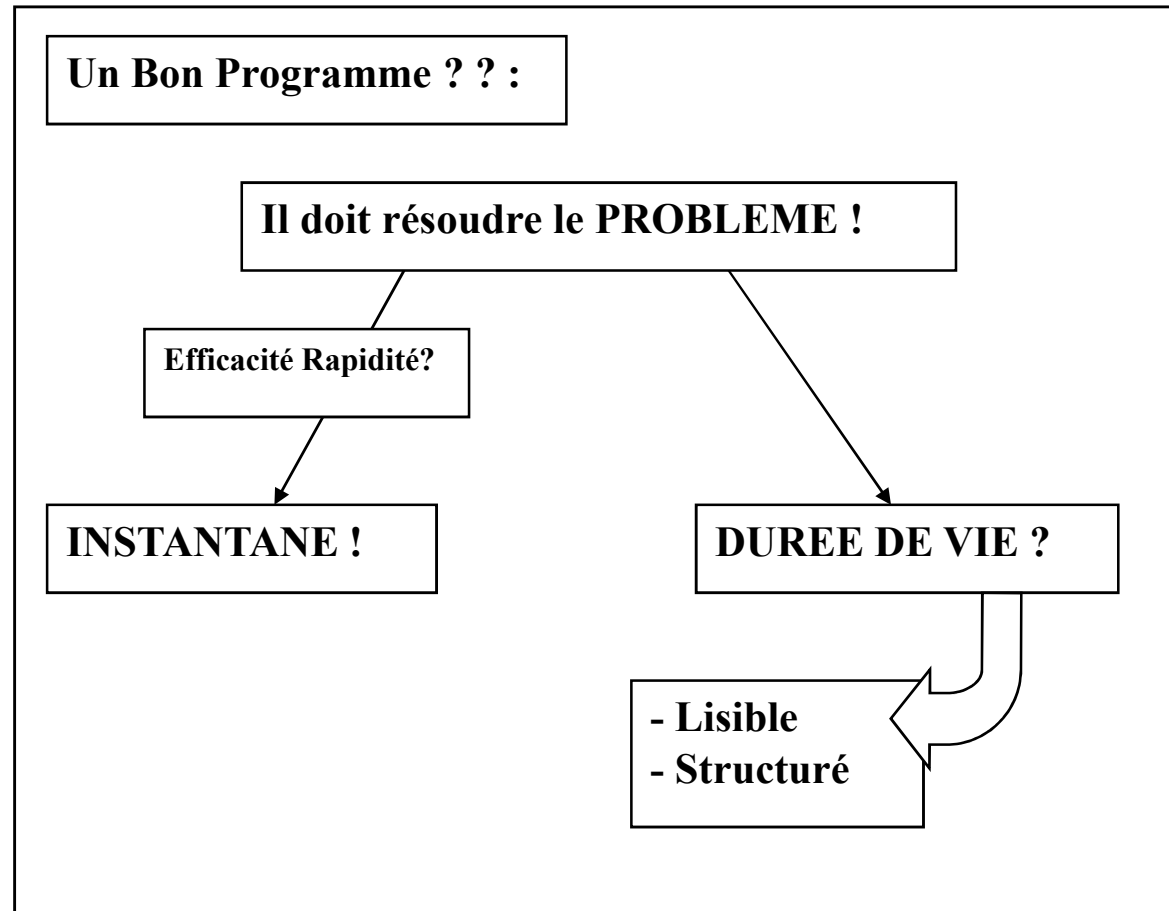

la programmation en langage C

Qu'est-ce qu'un bon programme?



Objectifs

- Être capable de bien programmer
- Comprendre les différentes constructions de la programmation en C
- Savoir programmer de manière modulaire

Qualités attendues d'un programme

- Clarté
- Simplicité
- Efficacité
- Modularité
- Extensibilité

Structure d'un programme C

```
#include <stdio.h>
#define DEBUT -10
#define FIN 10
#define MSG "Programme de démonstration\n"
```

```
int fonc1(int x);
void fonc2(int x);
```

```
void main()
```

```
{
    int i, x;
    /* début du bloc de la fonction main*/
    /* définition des variables locales */
    i = 0 ;
    x= fonc1(i) ;
    fonc2(i) ;
}
/* fin du bloc de la fonction main */
```

```
int fonc1(int x) {
    return x;
}
```

```
void fonc2(int x) {
    (x * x);
}
```

Directives du préprocesseur :
accès avant la compilation

Déclaration des fonctions

Programme principal

Définitions des fonctions

TYPE de la valeur de retour

"main" : Cela signifie "principale",
ses instructions sont exécutées.

int main(void)

```
begin {  
    /* corps du programme*/  
    declaration des Cstes et Var ;  
    instruction1 ;  
    instruction2 ;  
    ....  
} end
```

void main(void): La fonction main ne prend
aucun paramètre et ne retourne pas de valeur.

int main(void): La fonction main retourne une
valeur entière à l'aide de l'instruction return (0
si pas d'erreur).

Entre accolades "{" et "}" on
mettra la succession d'actions à
réaliser.(Bloc)

Indenter = lisibilité

Prenez l'habitude de respecter (au moins au début) les règles :

- une accolade est seule sur sa ligne,
- { est alignée sur le caractère de gauche de la ligne précédente,
- } est alignée avec l'accolade ouvrante correspondante,
- après { , on commence à écrire deux caractères plus à droite.

Fonctionnement :

- Taper et sauvegarder le programme,
- Compiler le programme,
- Exécuter le programme.

```
#include <Lib1.h>
#include <Lib2.h>
#define X 0;

int fonc1(int x);
float fonc2(char a);

int main(void)
{ /*main*/
    instruction;
    instruction;
    {
        instruction;
        {
            instruction;
        }
    }
    instruction;
} /* fin main*/
```

Préprocesseur

Le préprocesseur effectue un prétraitement du programme source avant qu'il soit compilé.

Ce préprocesseur exécute des instructions particulières appelées *directives*.

Ces directives sont identifiées par le caractère **#** en tête.

Inclusion de fichiers

#include <nom-de-fichier> /* répertoire standard */

#include "nom-de-fichier" /* répertoire courant */

La gestion des fichiers (**stdio.h**) /* Entrees-sorties standard */

Les fonctions mathématiques (**math.h**)

Taille des type entiers (**limits.h**)

Limites des type réels (**float.h**)

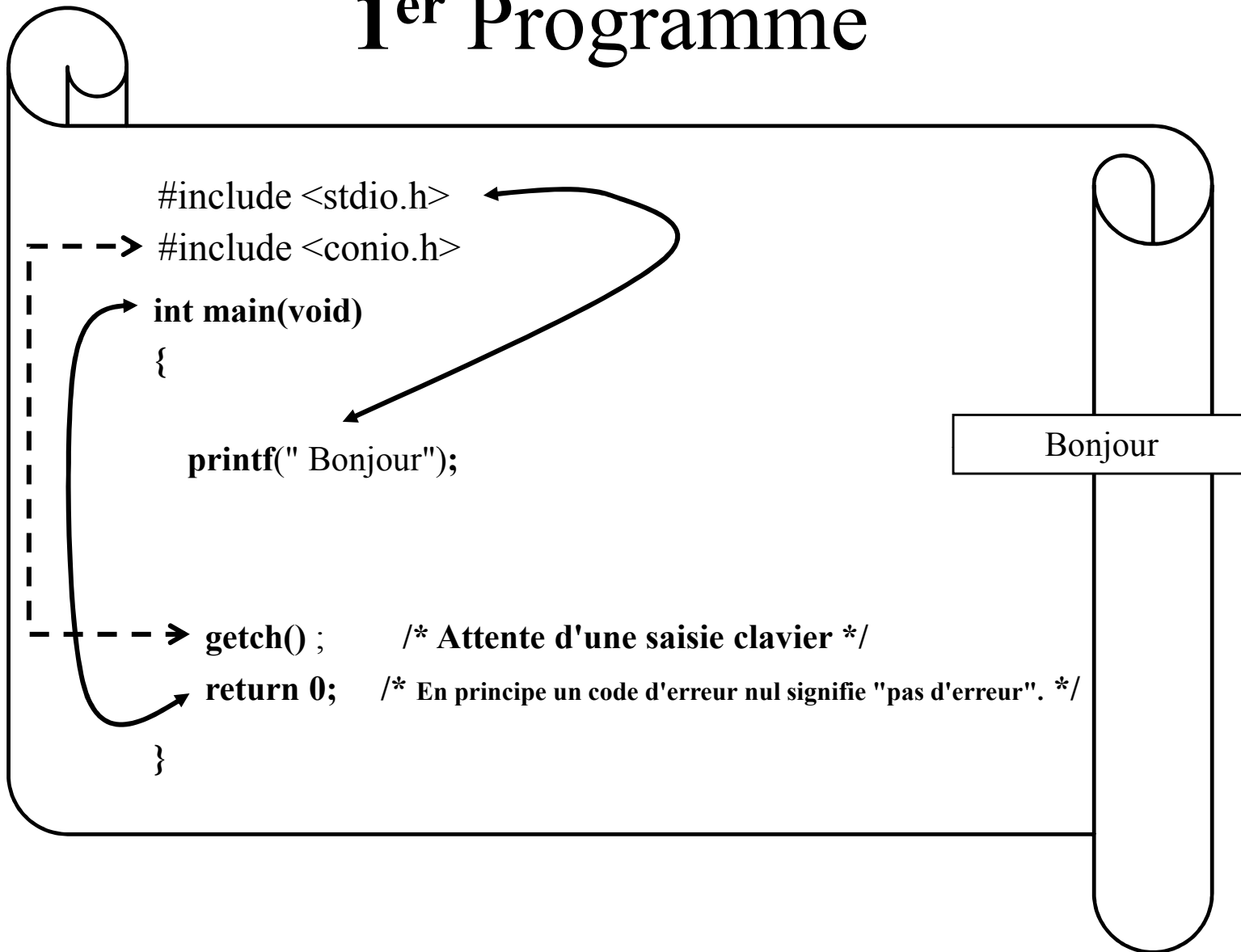
Traitement de chaînes de caractères (**string.h**)

Le traitement de caractères (**ctype.h**)

Utilitaires généraux (**stdlib.h**)

Date et heure (**time.h**)

1^{er} Programme



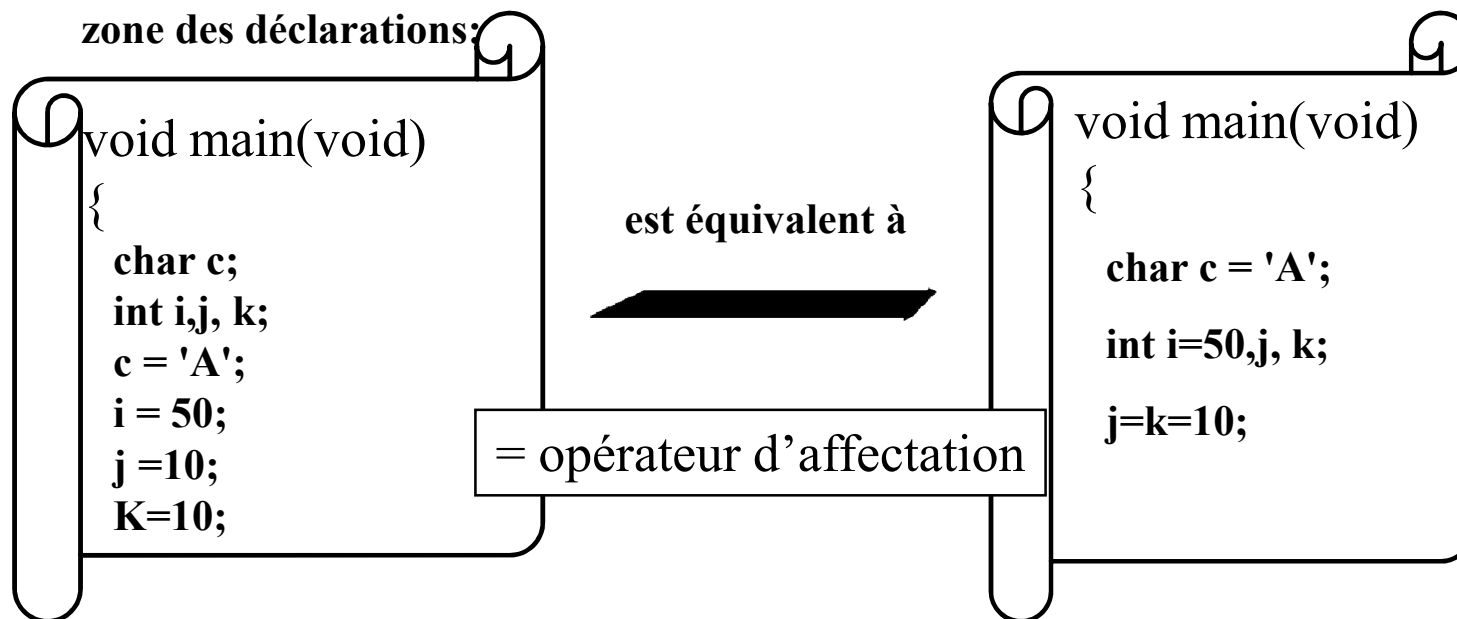
Variables : déclarations

Syntaxe : Type identificateur1, identificateur2, ...,..... ;

Exemple: char c1, c2, c3;
int i, j, var_ent;

Variables : initialisations

Les variables doivent être déclarées avant leur utilisation dans un début de bloc (juste après {}),



Types de base

4 types de base, les autres types seront dérivés de ceux-ci.

Type	Signification	Exemples de valeur	Codage en mémoire	Peut être
char	Caractère unique	'a' 'A' 'z' 'Z' '\n' 'a' 'A' 'z' 'Z' '\n' Varie de -128 à 127	1 octet	signed, unsigned
int	Nombre entier	0 1 -1 4589 32000 -231 à 231 +1	2 ou 4 octets	Short, long, signed, unsigned
float	Nombre réel simple	0.0 1.0 3.14 5.32 -1.23	4 octets	
double	Nombre réel double précision	0.0 1.0E-10 1.0 - 1.34567896	8 octets	long

Identificateurs

Les identificateurs nomment les objets C (fonctions, variables ...)

C'est une suite de lettres ou de chiffres.

Le premier caractère est obligatoirement une lettre.

Le caractère _ (souligné) est considéré comme une lettre.

Reconnaissance suivant les 31 premiers caractères.

Le C distingue les minuscules des majuscules.



Exemples :

abc, Abc, ABC sont des identificateurs valides et tous différents.

Identificateurs valides :

xx y1 somme_5 _position

Noms surface fin_de_fichier VECTEUR

Identificateurs invalides :

3eme commence par un chiffre

x#y caractère non autorisé (#)

no-commande caractère non autorisé (-)

taux change caractère non autorisé (espace)

Un identificateur ne peut pas être un mot réservé du langage :

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

RQ : Les mots réservés du langage C doivent être écrits en minuscules.

LES DECLARATIONS DE CONSTANTES

ie méthode: **déclaration** d'une variable, dont la valeur sera constante pour tout le programme

Exemple:

```
void main(  
{  
    const float PI = 3.14159;  
    const int JOURS =  
5;  
    float perimetre, rayon =  
8.7;  
    perimetre = 2*rayon*PI;  
    ....  
    JOURS = 3;  
    ....  
}
```

Le compilateur réserve de la place en mémoire (ici 4 octets).

/*ERREUR!*/ On ne peut changer la valeur d'une const.

Les opérateurs arithmétiques

- Le C propose les opérateurs suivants :

+ addition

- soustraction

* multiplication

/ division

% modulo (reste de la division entière)

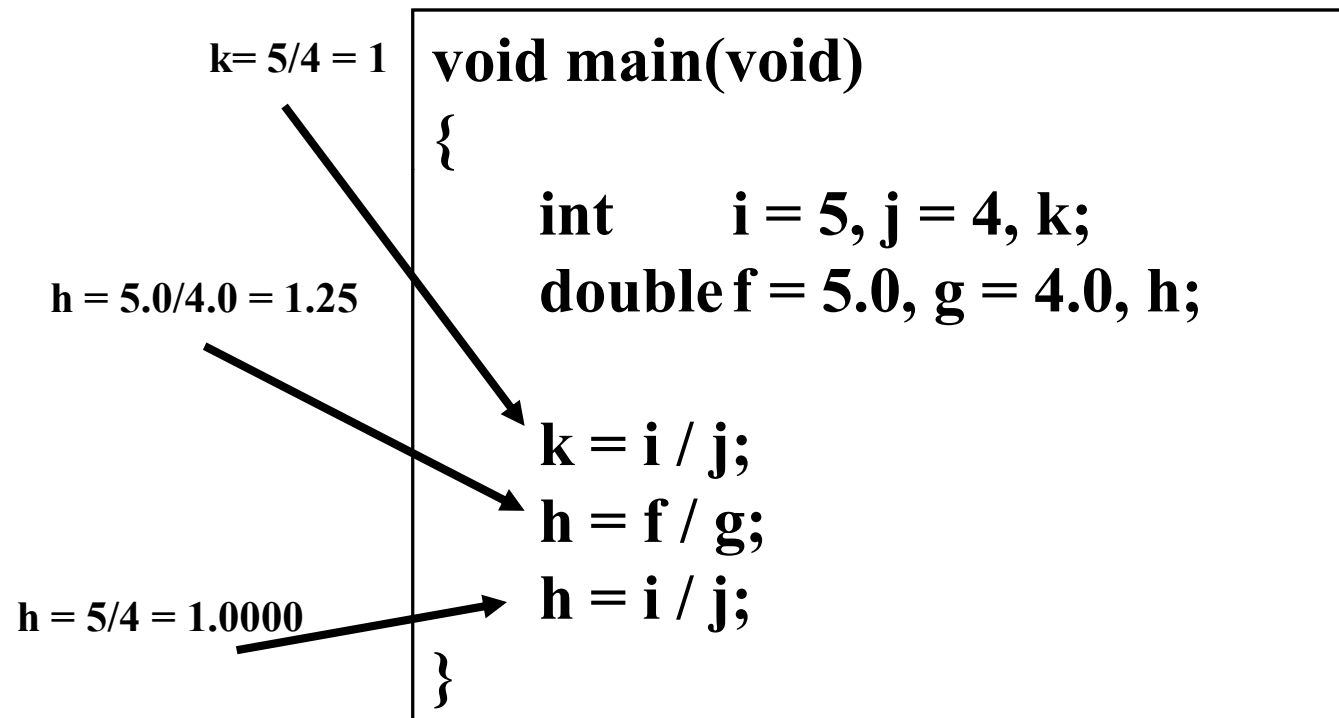
% ne peut être utilisé qu'avec des entiers

$7/2$ \longrightarrow 3

$7.0/2$
 $7/2.0$
 $7.0/2.0$ $\left. \vphantom{\begin{matrix} 7.0/2 \\ 7/2.0 \\ 7.0/2.0 \end{matrix}} \right\} \longrightarrow$ 3.5

Utiliser des opérateurs arithmétiques

Le compilateur considère le type des opérandes pour savoir comment effectuer les opérations



L'opérateur de conversion de type

L'opérateur de conversion de type, appelé **cast**, permet de modifier explicitement le type d'un objet.

On écrit: **(type) objet** Par exemple

```
Main( )  
{  
  int i=3, j=2;  
  float r;  
  r=(float)i/j;  
}
```

R va contenir 1,5



Les opérateurs de comparaison

<	plus petit
<=	plus petit ou égal
>	plus grand
>=	plus grand ou égal
==	égal
!=	différent



Le type booléen n'existe pas. Le résultat d'une expression logique vaut 1 si elle est

Les opérateurs logiques

&&	et
	ou (ou exclusif)
!	non

Réciproquement, toute valeur non nulle est considérée comme vraie et la valeur nulle comme fausse.



Exemple

```
int i;  
float f;  
char c;
```

```
i = 7;    f = 5.5;    c = 'w';
```

```
f > 5                =====> vrai (1)  
(i + f) <= 1        =====> faux (0)  
c == 'w'            =====> vrai (1)  
c != 'w'            =====> faux (0)  
c >= 10*(i + f)     =====> faux (0)  
(i >= 6) && (c == 'w') =====> vrai (1)  
(i >= 6) || (c == 119) =====> vrai (1)
```

!expr1 est vrai si expr1 est faux et faux si expr1 est vrai ;

expr1&&expr2 est vrai si les deux expressions expr1 et expr2 sont vraies et faux sinon. L'expression expr2 n'est évaluée que dans le cas où l'expression expr1 est vraie ;

expr1 || expr2 = (1) si expr1=(1) ou expr2=(1) et faux sinon.

L'expression expr2 n'est évaluée que dans le cas où l'expression expr1 est fausse.

Contractions d'opérateurs

- Il y a une famille d'opérateurs

`+=` `-=` `*=` `/=` `%=`
`&=` `|=` `^=`
`<<=` `>>=`

- Pour chacun d'entre eux

`expression1 op= expression2`

est équivalent à:

`(expression1) = (expression1) op (expression2)`

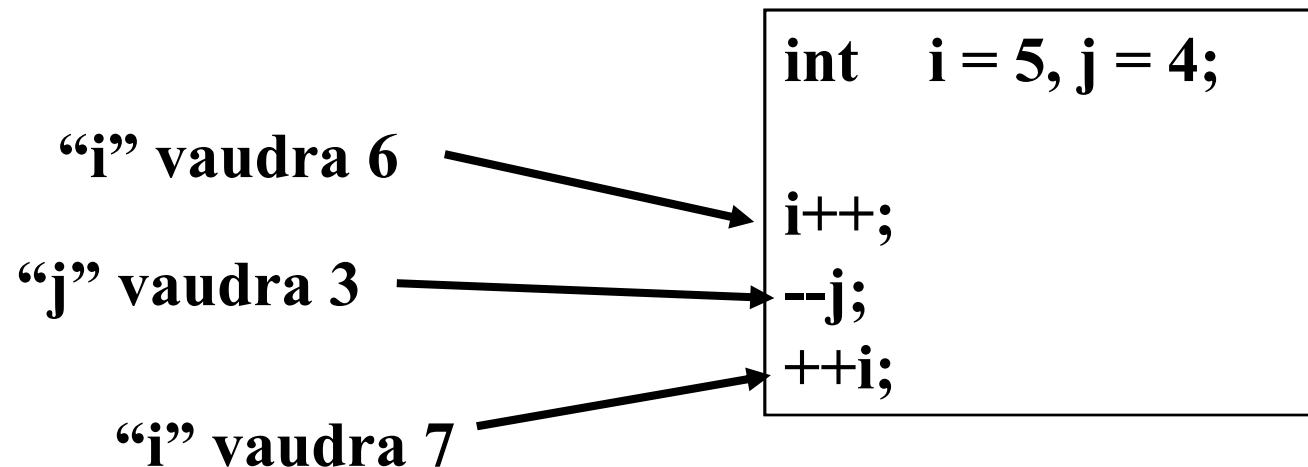
<code>a += 32;</code>	<code>f /= 9.2;</code>	<code>i *= j + 5;</code>
<code>a = a + 32;</code>	<code>f = f / 9.2;</code>	<code>i = i * (j + 5);</code>

Incrément et décrement

- C a deux opérateurs spéciaux pour incrémenter (ajouter 1) et décrementer (retirer 1) des variables entières

++ increment : $i++$ ou $++i$ est équivalent à $i += 1$ ou $i = i + 1$
-- decrement

- Ces opérateurs peuvent être préfixés (avant la variable) ou postfixés (après)



Préfixe et Postfixe

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

```
int main(void)
```

```
{
```

```
int i, j = 5;
```

```
i = ++j;
```

```
printf("i=%d, j=%d\n", i, j);
```

```
j = 5;
```

```
i = j++;
```

```
printf("i=%d, j=%d\n", i, j);
```

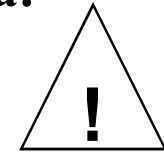
```
return 0;
```

```
}
```

équivalent à:

1. j++;

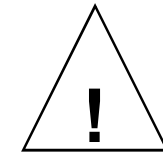
2. i = j;



équivalent à:

1. i = j;

2. j++;



i=6, j=6

i=5, j=6

Les fonctions d'E/S

La fonction **printf()** :

Librairie : **stdio.h.** **#include <stdio.h>**

Syntaxe : **int printf(const char *format [, arg [, arg]...]);**

Description : Permet l'écriture formatée (l'écran par défaut).

Exemple :

```
printf("Qu'il est agreable d'utiliser printf "  
"en\t C,\nlorsqu'on l'utilise \"proprement\".\n");
```

Résultat sur la sortie :

```
Qu'il est agreable d'utiliser printf en C,  
lorsqu'on l'utilise "proprement".
```

Les caractères précédés de \ sont interprétés comme suit :

```
\\ : caractère \  
\n : retour à la ligne  
\t : tabulateur.  
\" : caractère "  
\r : retour chariot
```


Les constantes de type caractère ont une valeur entière dans la table ASCII

```
char c1 = 'A',
```

```
c2 = '\x41'; /* représentation hexadécimale */
```

caractères	nom	symbole	code hexa	décimal
<code>\n</code>	newline	LF	10	
<code>\t</code>	tabulation	HT	9	
<code>\b</code>	backspace	BS	8	
<code>\r</code>	return	CR		13
<code>\f</code>	form feed	FF	12	
<code>\\</code>	backslash	5C	92	
<code>\'</code>	single quote	27	39	
<code>\"</code>	double quote	22	34	

La fonction **scanf()** :

Librairie : **stdio.h**. **#include <stdio.h>**

Syntaxe : `int scanf(const char *format [argument, ...]);`

Description : Lit à partir de stdin (clavier en principe), les différents arguments en appliquant le format spécifié.

Exemple : `scanf(" %d", &age);` /* lecture de l'âge, on donne l'adresse de age */



Format des paramètres passés en lecture et écriture.

`"%c"` : lecture d'un caractère.

`"%d"` ou `"%i"` : entier signé.

`"%e"` : réel avec un exposant.

`"%f"` : réel sans exposant.

`"%g"` : réel avec ou sans exposant suivant les besoins.

`"%G"` : identique à g sauf un E à la place de e.

`"%o"` : le nombre est écrit en base 8.

`"%s"` : chaîne de caractère.

`"%u"` : entier non signé.

`"%x"` ou `"%X"` : entier base 16 avec respect majuscule/minuscule.

1^{er} Programme

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
    int age;          /*déclaration d'une variable*/

    printf("Je te souhaite le bon"
           "jour aux TP\nEt je t"
           "e souhaite bon trav"
           "ail\n");

    printf("Quel est ton âge? ");
    scanf(" %d", &age);    /* lecture de l'âge, on donne l'adresse de age */
    printf("\nAlors ton age est de %d ans!\n",age);



    getch() ;           /* Attente d'une saisie clavier */
    return 0;           /* En principe un code d'erreur nul signifie "pas d'erreur". */
}
```

L'utilisation de & est indispensable avec scanf (valeur lue et donc modifiée), pas avec printf (valeur écrite et donc non modifiée).

!

Quel est ton âge ? 18





Alors ton age est de 18 ans!

printf  scanf 

Exemples	entrées	résultats
<pre>char c1, c2, c3; scanf(“%c%c%c”,&c1,&c2,&c3); scanf(“ %c %c %c”,&c1,&c2,&c3);</pre>	<p>a b c</p>	<p>c1=a c2=<espace> c3=b</p> <p>c1=a c2=b c3=c</p>
<pre>printf(“%kd”, n); affiche au moins k caractères à droite printf(“%f”,x); affiche le nombre réel en décimal avec 6 chiffre après la virgule printf(“%kf.J”,x);</pre>	<p>K=3 et n=25 K=2 et n= 355</p> <p>x= 1.5</p> <p>K=9 et x= 1.5 J=3</p>	<p>Affiche 25_ Affiche 355</p> <p>Affiche 1.500000</p> <p>Affiche _____.1500</p>

Affichages et saisies

Librairie : stdio.h

Fonction	Syntaxe	Description
printf	printf(const char *format [, arg [, arg]...]);	Écriture formatée  sortie standard
scanf	scanf(const char *format [, arg [, arg]...]);	Lecture formatée  entrée standard
putchar	putchar(int c);	Écrire le caractère c 
getchar getch	getchar(); getch(); <conio.h>	Lecture d'un caractère 
puts gets	*puts(char *s); *gets(char *s);	Ecriture/Lecture d'une chaîne de caractères, terminée par \n
sprintf	sprintf(char *s, char *format, arg ...);	Ecrit dans la chaîne d'adresse s.
sscanf	sscanf(char *s, char *format, pointer ...);	Lit la chaîne d'adresse s.