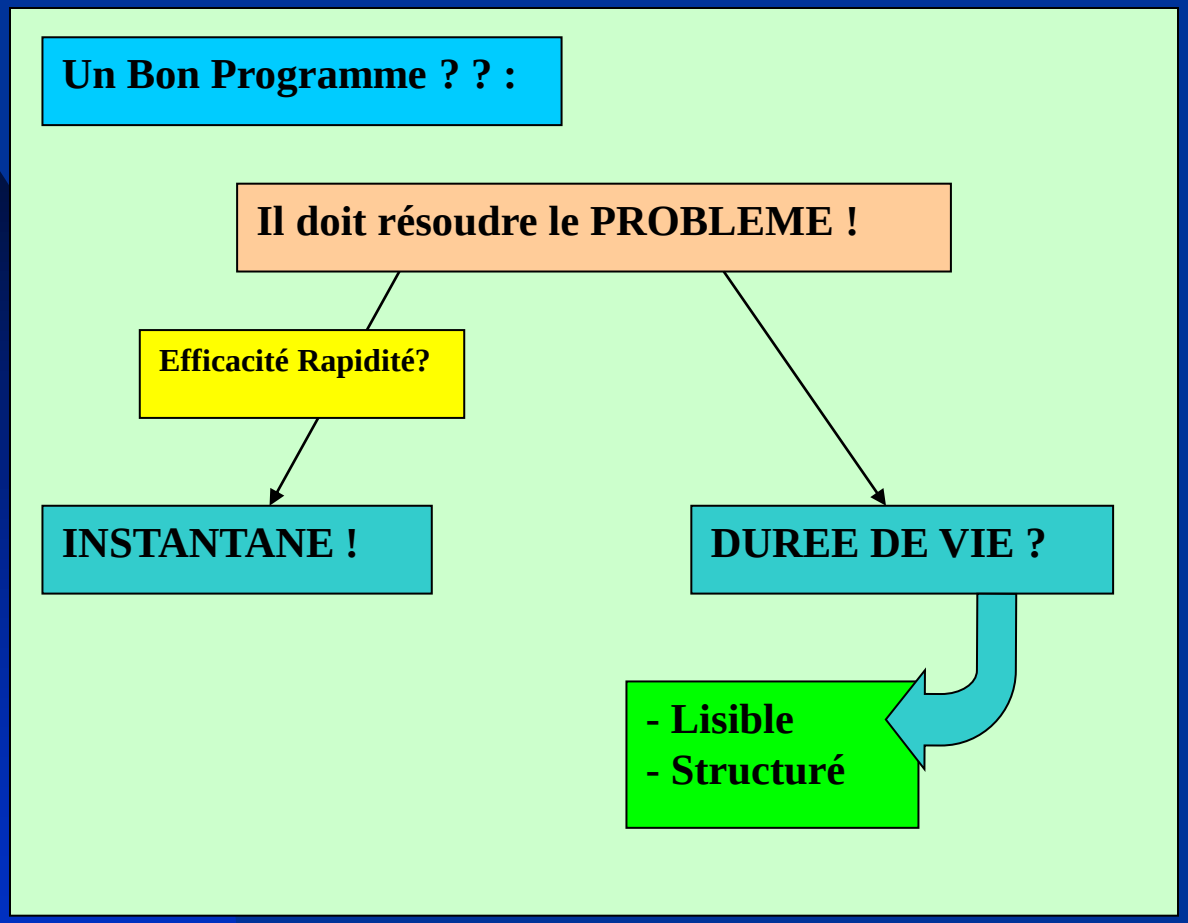
Syntaxe de base du langage C

Qu'est-ce qu'un bon programme?



Qualités attendues d'un programme

- Clarté
- Simplicité
- Efficacité
- Modularité
- Extensibilité

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

TYPE de la valeur de retour

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

```
int main(void)
{
/* corps du programme*/
declaration des Cstes et Var ;
instruction1 ;
instruction2 ;
....
}
```

begin

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).

int main(int argc, char *argv[]): On obtient alors des programmes auxquels on peut adresser des arguments au moment où on lance le programme.

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

Structure d'un programme C

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

Directives du préprocesseur : accès avant la compilation

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

Déclaration des fonctions

```
void main()
{
  int i;

  i = 0 ;
  fonc1(i) ;
  fonc2(i) ;
}

int fonc1(int x) {
   return x;
}

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

```
/* début du bloc de la fonction main*/
/* définition des variables locales */

/* fin du bloc de la fonction main */
```

Programme principal

Définitions des fonctions

Indenter = lisibilté

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**)

1er Programme

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

   printf(" BTS GI ");



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

}
```

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 entiére dans la table ASCII

char c1 = 'A',

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

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

1er 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!
```

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{),

zone des déclarations:

```
void main(void)
{
  char c;
  int i,j, k;
  c = 'A';
  i = 50;
  j =10;
  K=10;
```

est équivalent à

= opérateur d'affectation

```
void main(void)
{
  char c = 'A';
  int i=50,j, k;
  j=k=10;
```

Cette règle s'applique à tous : char, int, float ...

scanf

Exemples	entrées	résultats
char c1, c2, c3; scanf(‘’%c%c%c‘’,&c1,&c2,&c3);	a b c	c1=a c2=<espace> c3=b
scanf(‘’ %c %c %c‘’,&c1,&c2,&c3);		c1=a c2=b c3=c
char c; int i; float x; scanf(‘’%2d %5f %c‘’,&i,&x,&c);	12 123.567 r	i=12 x=123.5 c=6

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.

LES DECLARATIONS DE CONSTANTES

1ere méthode: **définition d'un symbole** à l'aide de **la directive** de compilation **#define.**

Exemple:

```
#define   PI   3.14159
void main()
{
  float perimetre,rayon = 8.7;
  perimetre =  2*rayon*PI;
  ....
}
```

Le compilateur ne réserve pas de place en mémoire

Syntaxe : `#define identificateur texte(valeur)`


```
#define  TAILLE          100
#define MAXI   (3 * TAILLE + 5)
#define nom_macro(identif_p1 , ... ) texte
#define SOMME(X,Y)  X+Y
#define MULTIP(A,B)  (A)*(B)
```

Les identificateurs s'écrivent traditionnellement en majuscules, mais ce n'est pas une obligation.

LES DECLARATIONS DE CONSTANTES

2eme 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.

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 opérateurs arithmétiques

- Le C propose les opérateurs suivants :

+	addition
-	soustraction
*	multiplication
/	division
%	modulo (reste de la division entière)

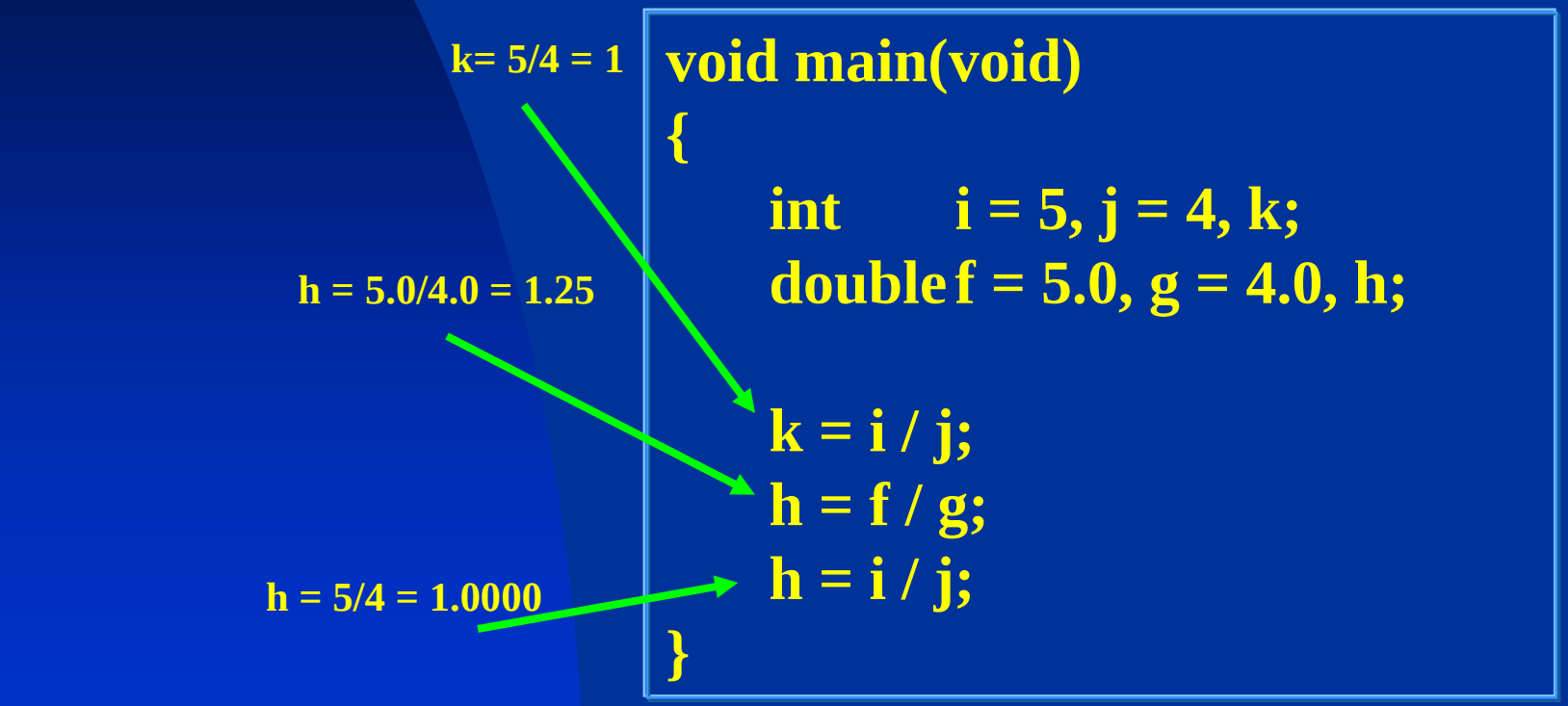
% ne peut être utilisé qu'avec des entiers

7/2 → 3

7.0/2
7/2.0
7.0/2.0
} → 3.5

Utiliser des opérateurs arithmétiques

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



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 vraie et 0 sinon.

Les opérateurs logiques

&&	**et**
||	**ou (non 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)

```
a += 32;
```
```
a = a + 32;
```

```
f /= 9.2;
```
```
f = f / 9.2;
```

```
i *= j + 5;
```
```
i = i * (j + 5);
```

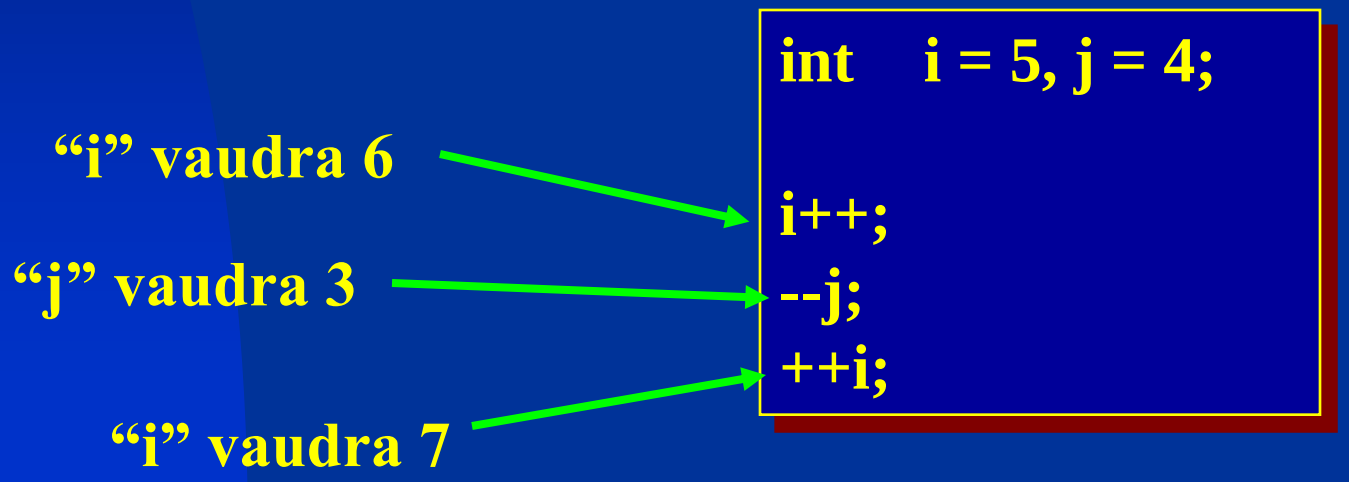
Incrément et décrement

- C a deux opérateurs spéciaux pour incrémenter (ajouter 1) et décrémenter (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 Conversions de types

Le langage C permet d'effectuer des opérations de conversion de type.
On utilise pour cela l'opérateur de "cast" ().

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int i=0x1234, j;
        char d,e;
        float r=89.67,s;
        j = (int)r;
        s = (float)i;
        d = (char)i;
        e = (char)r;
        printf("Conversion float -> int: %5.2f -> %d\n",r,j);
        printf("Conversion int -> float: %d -> %5.2f\n",i,s);
        printf("Conversion int -> char: %x -> %x\n",i,d);
        printf("Conversion float -> char: %5.2f -> %d\n",r,e);
        printf("Pour sortir frapper une touche ");
        getch();        // pas getchar
}
```

```
Conversion float -> int: 89.67 -> 89
Conversion int -> float: 4660 -> 4660.00
Conversion int -> char: 1234 -> 34
Conversion float -> char: 89.67 -> 89
Pour sortir frapper une touche
```