Langage C

Connaissances minimum au développement logiciel sur micro-contrôleurs PIC en C ANSI

Christian Dupaty
Académie d'Aix-Marseille

Pourquoi un langage évolué ?

- **Le langage C est :**
  - **Evolué** : Le code est indépendant du processeur utilisé
  - **Typé** : Un type est l'ensemble des valeurs que peut prendre une variable
  - **Modulaire et structuré** :Tout programme est décomposable en tâches simples qui seront regroupées sous forme de fonctions qui regroupérs de façon cohérente en tâches plus complexes (structurés) formeront le programme.

```
#include
<stdio.h>

main()
{ puts(" Bonjour
à tous ");
}
```

Bibliothèques en C (texte) *.c

Fichiers header *.h

Bibliothèques pré-compilées (fichiers objet)

Fichier source C contenant la fonction main

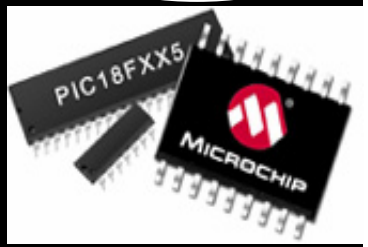
Préprocesseur
Remplace les #define et effectue les #include

Compilateur C
Transforme le fichier C en un fichier objet (code machine), les fonctions pré-compilés sont déclarées dans les fichiers *.h

Editeur de lien
LINKER
Lie (donne des adresses aux fonctions) tous les fichiers objets et crée un fichier exécutable

Fichier de symboles pour debug

Programme exécutable



FLUX DE DONNEES D'UN COMPILATEUR C

christian.dupaty@ac-aix-marseille.fr

Premier programme

```
#include <p18f452.h>
#define duree 10000
char c;
float pht;
void tempo(unsigned int count);
void main(void)
{
        PORTB = 0x00;
        TRISB = 0x00;
        while(1) {
                PORTB++;
                tempo(duree);
                }
}

void tempo(unsigned int compte)
{
        while(compte--);
}
```

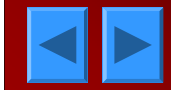
Equivalence : Le pre-processeur remplacera tous les duree par 1000

Déclaration de deux variables réelles

Entrée du programme principal

Représentation des nombres :
12 codé en décimal représente 12
0xC codé en hexadécimal représente 12
0b00001100 codé en binaire représente 12

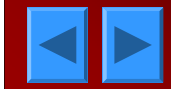
Fonction ou sous programme



christian.dupaty@ac-aix-marseille.fr

Les types de données

Type	Longueur	Domaine de valeurs
signed char	8 bits	-128 à 127
unsigned char	8 bits	0 à 255
signed int	16 bits	-32768 à 32767
unsigned int	16 bits	0 à 65535
long	32 bits	-2,147,483,648 à 2,147,483,647
unsigned long	32 bits	0 à 4,294,967,295
float	32 bits	3.4 * (10**-38) à 3.4 * (10**+38)
double	64 bits	1.7 * (10**-308) à 1.7 * (10**+308)



christian.dupaty@ac-aix-marseille.fr

Exemples de déclaration

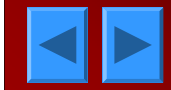
```
char a,b,c ;
int table[100] ;
char tableau[ ]={10,0x1c,'A',55,4}
char *chaine= "bonjour" ;
char *p ;
```

trois caractères

tableau de 100 entiers

chaîne de 8 caractères (finie par 0)

le symbole ***** désigne un **pointeur** sur un type défini
p est un pointeur sur des caractères



christian.dupaty@ac-aix-marseille.fr

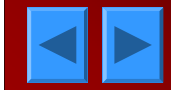
Equivalences

directive #define

Les équivalences sont remplacées par leur valeur par le pré-processeur

#define pi 3.14

#define fruit pomme



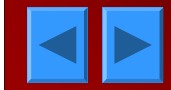
christian.dupaty@ac-aix-marseille.fr

Constantes

- **elles sont rangées dans la ROM et ne sont donc pas modifiables.**
- **const int i=16569, char c=0x4c ;**
- **const float pi=3.14;**
- **near** indique une adresse sur 16bits au contraire de **far** sur 21 bits ou plus

Le mot réservé « const » définit une constante

*Certains compilateurs utilisent **ram rom** pour définir l'emplacement de stockage ex : ram const int i=16569;*



christian.dupaty@ac-aix-marseille.fr

Variables

- elles sont rangées dans la RAM
  char a,b=28,c='A' ;

  « ***volatile*** **»** déclare une variable modifiable par l'environnement (un port // en entrée par exemple)
  ex : ***volatile*** **char a;**

VARIABLES LOCALES ET GLOBALES

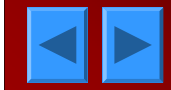
- GLOBLALES ← **Déclarées en dehors d'une fonction toujours statiques**
- LOCALES ← **Déclarées dans une fonction**

soit à une adresse fixe (statique)

static char var;

soit dans une pile LIFO (automatique)

char var;



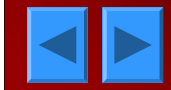
christian.dupaty@ac-aix-marseille.fr

Opérateurs unaires

()	Appel de fonction	
[]	Indice de tableau	tableau[3]=5;
!	Négation logique (NOT)	
~	Complément binaire bit à bit	b=~a
-	Moins unaire	b=-a;
+	Plus unaire	b=+a;
++	Pré ou postincrément	b=a++;
--	Pré ou postdécrément	b=a--;
&	Adresse de	b=&a;
*****	Indirection	b=*a;

Opérateurs binaires

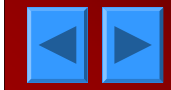
*	Multiplication	c=a*b;
/	Division	c=a/b;
+	Plus binaire	c=a+b;
-	Moins binaire	c=a-b;
<<	Décalage à gauche	c=a<<b;
>>	Décalage à droite	c=a>>b;
&	ET entre bits	c= a & b;
^	OU exclusif entre bits	c= a ^b;
|	OU entre bits	c= a|b;



christian.dupaty@ac-aix-marseille.fr

TESTS

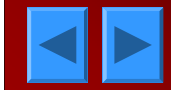
<	Strictement inférieur	if (a < b)
<=	Inférieur ou égal	if (a >= b)
>	Strictement supérieur	if (a > b)
>=	Supérieur ou égal	if (a >= b)
==	Egal	if (a ==b)
!=	Différent	if (a != b)
&&	ET logique	if ((a==5) && (b==2))
||	OU logique	if ((a==5) ||(b==2))
?:	Condition z=(a>b)?a:b	(Si a>b a z=a sinon z=b)



christian.dupaty@ac-aix-marseille.fr

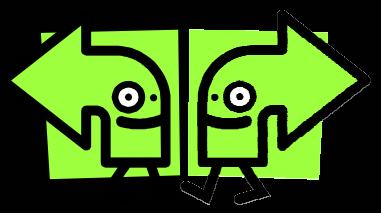
Affectations et Auto-affectations

=	Affectation simple	a=b;
***=**	Affectation produit	a*=2 (a=a*2)
/=	Affectation quotient	a/=2 (a= a/2)
%=	Affectation reste	a%=2 (a= le reste de a/2)
+=	Affectation somme	a+=2 (a=a+2)
-=	Affectation différence	a-=2 (a=a-2)
&=	Affectation ET entre bits	a&=5 (a=a&5)
^=	Affectation OU EX entre bits	a^=5 (a=a^5)
|=	Affectation OU entre bits	a|==5 (a=a|=5)
<<=	Affectation décalage gauche	a<<=5 (a=a<<5)
>>=	Affectation décalage droite	a>>=5 (a=a>>5)



christian.dupaty@ac-aix-marseille.fr

Boucle FOR



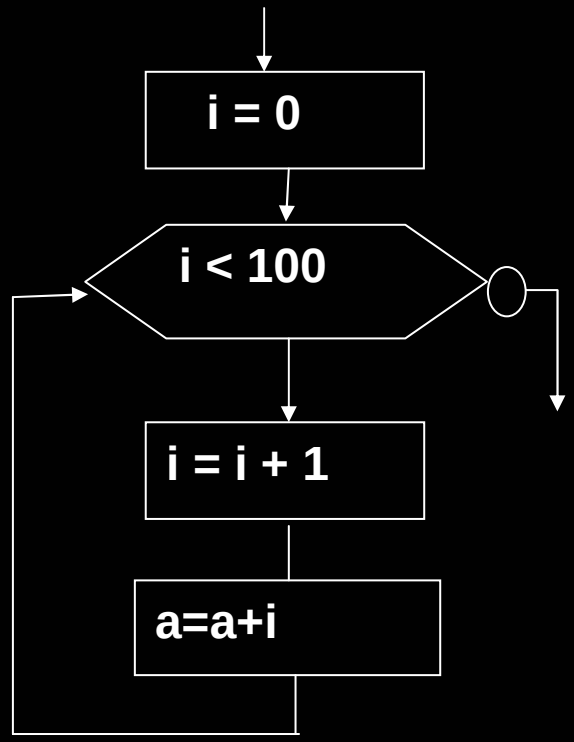
```
int a,i;


for (i=0 ;i<100 ;i++)
 {
   a=a+i;
}
```

Condition de départ

Condition pour rester dans la boucle

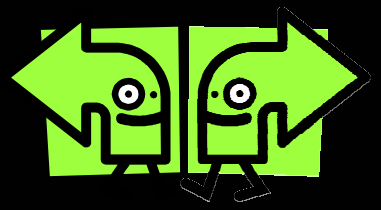
Évolution d'une variable



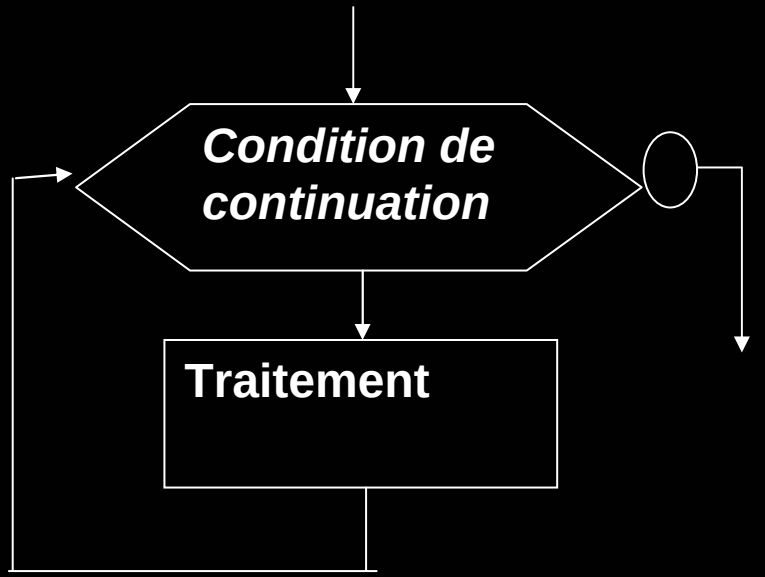

```
char i=20 ;
for (;i<100 ;i++) a=a+i; /* Pas de condition de départ*/
for( ;;) ;                /*une boucle sans fin non standard*/
```

christian.dupaty@ac-aix-marseille.fr

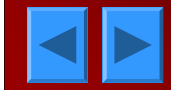
Boucle WHILE



```
i=0;
while (i<100)
   {
      a=a+i;
      i++ ;
   }
```

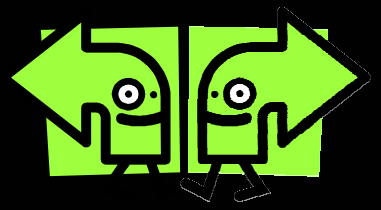


Si la condition de boucle est fausse au départ la boucle n'est jamais effectuée

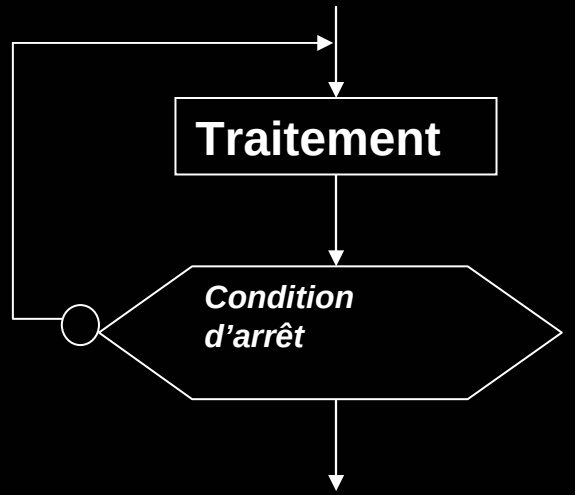


christian.dupaty@ac-aix-marseille.fr

Boucle DO WHILE



```
do
   {
      a=a+i;
      i++ ;
   }
while (i<100)
```



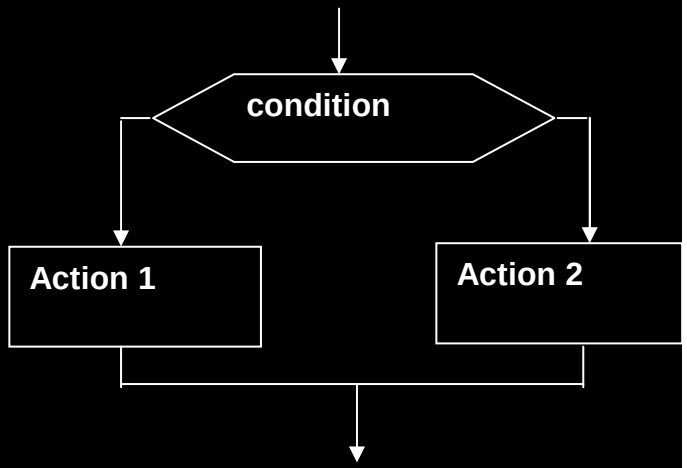
la boucle est toujours effectuée au moins une fois

christian.dupaty@ac-aix-marseille.fr

Branchement : if - else

Condition du test : ==, <,>,<=,>=, !=,&&,|| …

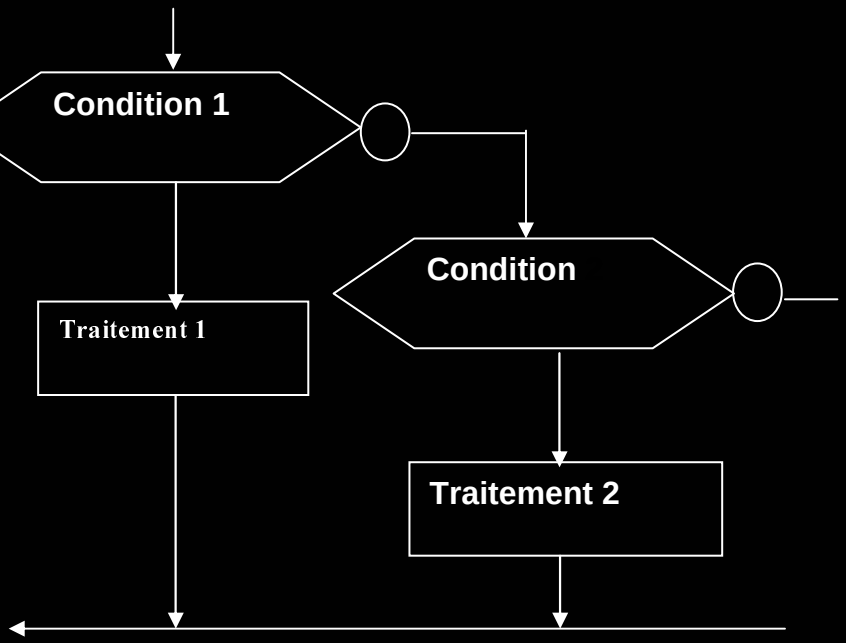
```
int calc(char c)
{
        if (c=='+') s=a+b; else
        if (c=='-') s=a-b; else
        if (c=='/') s=a/b; else
        if (c=='*') s=a*b;
        return(s);
}
```



christian.dupaty@ac-aix-marseille.fr

Brachement switch-case

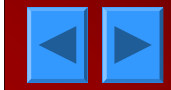
```
int calc(char c)
{
switch (c )
    {
    case '+' : return (a+b);
    case '-' : return (a-b);
    case '*' : return (a*b);
    case '/' : return (a/b);
    default : return(0);
    }
}
```



christian.dupaty@ac-aix-marseille.fr

Contrôle des boucles

- **break** permet de sortir de la boucle en cours (for, while, do while, switch
- **continue** permet de sauter directement à l'itération suivante d'une boucle
  for(i=0 ;i<100 ;i++)
  { if (i<50) continue else a=a+i;}
- **exit** permet de quitter directement le programme (inutile sur micro contrôleur)



christian.dupaty@ac-aix-marseille.fr

Pointeurs et données

p: pointeur en 0x8000-0x8001

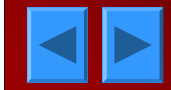
a: char en 0x9000

b: char en 0x9001

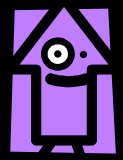
Exécution de b=*p;

Execution de p=p+1;

ADRESSES	DONNEES
0x8000	0x90
0x8001	**0X01**
0x8002	??
…..	
0x9000	0xAA
0x9001	**0XAA**
0x9002	??



christian.dupaty@ac-aix-marseille.fr



POINTEURS

Ce sont des **variables qui contiennent l'adresse d'une variable**,
Sur micro-contrôleur PIC18 un pointeur est une valeur sur 16bits ou 20bits

Un pointeur est déclaré par une * précédée du type de donnée pointée
Le signe & devant une donnée indique l'adresse de celle ci et sa valeur.

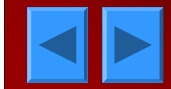
char *p ; déclare un pointeur p sur un caractère

float *f ; déclare une pointeur sur un réel.

char *fonction(void) déclare une fonction qui retourne un pointeur sur un caractère

void (*fonction) (void) déclare un pointeur sur une fonction

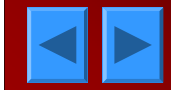
void (*fonction) (void) = 0x8000 crée un pointeur sur une fonction en 8000



christian.dupaty@ac-aix-marseille.fr

manipulation de pointeurs

- **int a=1,b=2,c ;**/*trois entiers */
- **int *p1,*p2 ;** /*deux pointeurs sur des entiers*/
- **p1=&a ;** /*p1 contient l'adresse de a*/
- **p2=p1 ;** /*p2 contient l'adresse de a*/
- **c=*p1 ;** /*c égale le contenu de l'adresse pointé par p1 donc c=a*/
- **p2=&b ;** /*p2 pointe b*/
- ***p2=*p1**
  /*la donnée à l'adresse pointé par p2 est placée dans l'adresse pointé par p1, cela revient à donc recopier a dans b*/



christian.dupaty@ac-aix-marseille.fr

exemple d'utilisation des pointeurs : la fonction echange :

echange(x,y);

```
void echange(int i ,int j)
{
int k;
k=i;
i=j;
j=k;
}
```

i et j sont permutés mais PAS x et y

echange(&x,&y);

```
void echange(int *i ,int *j)
{
int k
k=*i ;
*i=*j ;
*j=k ;
}
```

Les contenus des adresses sont permutés, i et j pointant x et y

christian.dupaty@ac-aix-marseille.fr

TABLEAUX

Un tableau est un regroupement de variables de même type

int chiffres[ ]={10,11,12,13,14,15,16,17,18,19}

int TAB[20]={1,12,13}

chiffre[0]=10, et
chiffre[3]=13

TAB[3] à TAB[19] = 0

TAB est l'adresse de début du tableau

TAB représente &TAB[0]
TAB[0] représente *TAB
TAB+i = &TAB[i]
TAB+1 pointera la donnée suivante

char TAB[2][3]={{1,2,3},{4,5,6}}

TAB[1][1]=5

christian.dupaty@ac-aix-marseille.fr

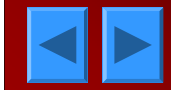
Chaînes de caractères

Ce sont des tableaux de caractères finissant par 0,
une chaîne est entourée de "

```
char message[ ]= "bonjour";
char message[ ]={‘b’,’o’,’n’,’j’,’o’,’u’,’r’,\0} ;

char *p= message ;
while (*p !=0) putchar(*p++) ;
```

← La fonction puts(char *string);



christian.dupaty@ac-aix-marseille.fr

CAST – transformation de type

CAST automatiques :

char -> int -> long -> float -> double

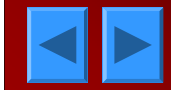
signed -> unsigned

float x ; int a=5 ;

x=(float)a ; → x=5.0

float x=5.6 ; int a ;

a=(int)x ; → a=5



christian.dupaty@ac-aix-marseille.fr

Initialisation d'un pointeur à une adresse absolue

```
#define PORTA *(unsigned char *)(0xFE0)
```

0xFE0 est un pointeur sur un caractère (CAST)

PORTA est le contenu de cette adresse

ex: var=PORTA

PORTA=var

christian.dupaty@ac-aix-marseille.fr

STRUCTURES

- Une structure est composée de variables de types différents.

Nouveau type (facultatif)

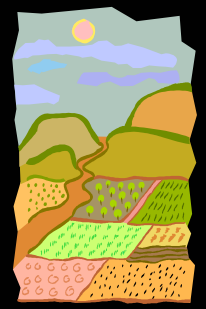
- struct identite { char nom[30];
  int age;
  } jean,jacques,groupe[20];

jean.nom=« bonnot »;

a=jean.age;

christian.dupaty@ac-aix-marseille.fr

Structures-Champs de bits



```
struct {
        unsigned RB0:1;
        unsigned RB1:1;
        unsigned RB2:1;
        unsigned RB3:1;
        unsigned RB4:1;
        unsigned RB5:1;
        unsigned RB6:1;
        unsigned RB7:1;
      } PORTBbits ;
```

Utilisation :

PORTBbits.RB2=1;

A=PORTBbits.RB2;

christian.dupaty@ac-aix-marseille.fr

STRUCTURES - UNION

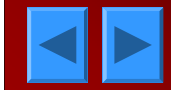
- Dans une UNION les champs partagent les mêmes adresses.

```
volatile near union  {
        struct {
           unsigned RE0:1;
           unsigned RE1:1;
           unsigned RE2:1;
           unsigned RE3:1;
           unsigned RE4:1;
           unsigned RE5:1;
           unsigned RE6:1;
           unsigned RE7:1;
            } ;
        struct {
           unsigned ALE:1;
           unsigned OE:1;
           unsigned WRL:1;
           unsigned WRH:1;
           unsigned :3;
           unsigned CCP2:1;
            } ;
        struct {
           unsigned AN5:1;
            } ;
    } PORTEbits ;
```

PORTEbits.RE0,
PORTEbits.ALE
PORTEbits.AN5
désignent le même bit du même registre

Les bits 4,5,6 ne sont pas déclarés

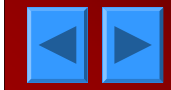
Seul le bit 0 est déclaré



christian.dupaty@ac-aix-marseille.fr

Bibliothèques standards C ANSI

- **ctype.h** trouver les types ex: isdigit (chiffre) ou islower
- **limits.h** indique les limites des types
- **string.h** traitement sur les chaînes
- **math.h** fonctions mathématiques
- **stdlib.h** conversion de chaines (atoi atof) génération d'un nombre aléatoire (rand, srand) allocation dynamique de mémoire (malloc, calloc), tri (qsort)
- **time.h** fonctions liée à l'heure et à la génération de nombre aléatoires



christian.dupaty@ac-aix-marseille.fr

Ecrire court en C

```
void strcpy(char *s, char *t)
{   int i;
    i=0;
    do
        {s[i] =t[i]
        i++;
        }
    while (s[i-1] != '\0');
}
```

```
void strcpy(char *s, char *t)
{
while((*s++=*t++) != '\0') ;
}
```

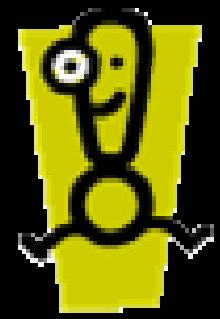
```
void strcpy(char *s, char *t)
{   while((*s=*t) != '\0')
    {       s++ ;
            t++ ;
    }
}
```

```
void strcpy(char *s, char *t)
{
while (*s++=*t++)  ;
}
```

D'après C ANSI de K&R Masson

christian.dupaty@ac-aix-marseille.fr

Entrées/sorties caractères (C ANSI)



- **Les deux fonctions d'E/S sur caractère**

Elle sont définies pour le matériel support des communications

- **char getchar(void)**

← Lit un caractère en entrée (CLAVIER par exemple)

- **int putchar(char)**

← Envoie un caractère (LCD par exemple)

christian.dupaty@ac-aix-marseille.fr

Entrées/sorties Chaînes (C ANSI)

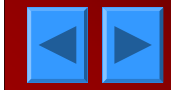
A partir de getchar et putchar

affiche une chaîne de caractères

- **void puts(char *chaîne) ;**

- **char *gets(char *chaîne) ;**

saisie une chaîne de caractère finie par un RC et retourne un pointeur sur le premier caractère de cette chaîne

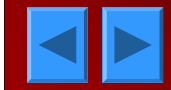


christian.dupaty@ac-aix-marseille.fr

Formats sur printf (C ANSI)

- %c (char)
- %s (chaîne de caractères, jusqu'au \0)
- %d (int)
- %u (entier non signé)
- %x ou X (entier affiché en hexadécimal)
- %f (réel en virgule fixe)
- %p (pointeur)
- % (pour afficher le signe %)

\ n nouvelle ligne
\ t tabulation
\ b backspace
\ r retour chariot
\ f form feed
\ ' apostrophe
\ \ antislash
\ " double quote
\ 0 nul



christian.dupaty@ac-aix-marseille.fr

Merci de votre attention

Exercez vous,

un langage s'apprend par la pratique

Compilateur C/C++ gratuits :

- *DOS : TURBO C/C++ V1.01* ***www.borland.fr***
- *WIDOWS : DEV-C++* ***www.bloodshed.net***

christian.dupaty@ac-aix-marseille.fr