Programmation C

I- Type de données : (variable & constante)

Type

Nom	Type	Valeur
Entier	Réel	caractère
• int 2o	• float 4o	• char 1o (-128 à 127)
• long 4o	• double 8o	• unsigned char 1o (0 à 255)
• unsigned int 2o (0 à 65535)	• long double 16o	

Booléen • bool (0 ou 1) 1o

Déclaration d'une donnée :

* Constante :

const Type Nom_constant = valeur ;

* Variable :

Type Nom_variable ;

Affectation :

Nom_variable = valeur ;

II - Les opérateurs :

- + - / * et % ← opérateurs arithmétique (% : reste de la division entière)
- <, >, <=, >=, == (égalité), != (différent) opérateurs relationnels
  - ↳ le résultat est un entier (0 → faux, 1 → vrai)

Commentaire : /* commentaire */

- && (And logique) op1 && op2 est vrai (1) / faux (0)
- || (or logique) op1 || op2 est faux ssi op1 et op2 sont faux
- ! (not logique)

(les opérateurs logiques)

Bit = 0/1 ; ou exclusif (^) ; Non (~)

Incrémentation (++)

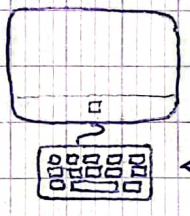
X = X+1 ⟷ X += 1

X++ ⟷ ++X

Décrémentation (--)

X = X-1 ⟷ X -= 1

X-- ⟷ --X



Printf() ← afficher la valeur d'une expression sur l'écran

Scanf() → Demander à l'utilisateur de fournir l'information (chaque information stockée dans une variable)

Printf

Printf ("Texte...., var, const, expr, ...) ;

- Le texte à afficher ↵
- Les spécificateurs de format
- Caractère d'échappement

variables, constantes ou expressions dont les valeurs on affiche

exp : Printf ("Maroc") } affiché sur l'écran le message "Maroc"

Printf ("A = %d", A) } affiché sur l'écran le contenu des variable A, X et Y

Printf ("les coordonnées sont : %f, %f", X, Y)

scanf

scanf ("format_1,...", &var_1,...);

spécificateur de format — adresse — Nom de variable

exp:
- scanf ("%d", &x); demander la valeur de x entier
- scanf ("%c", &y); ——— y (caractère)
- scanf ("%d %d %d", &jour, &mois, &année); demander le jour, le mois et l'année

(sans espace)

Les spécificateur de format:

"%d"	→ int	"\n"	Retour à la ligne
"%ld"	→ long	"\t"	Tabulation horizontale
"%f"	→ float ou double	"\'"	Affiche une apostrophe
"%c"	→ char	"\""	——— un guillemet
		"\\"	——— un antislash
		"%%"	——— un %

La structure d'un programme C

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
```
} Appel des bibliothèque

```
int main() {
    printf("bonjour");
    return 0;
}
```
← Fonction Principale: partie traitement

- la fonction pow (puissance)

exp:
```
#include <math.h>
int main() { int x, y, z;
    x = 4;
    y = 3;
    z = pow(x, y);
    printf("%d", z);
    return 0;
}
```

III - Les structures de Contrôles:

* if et else imbriqués:

```
if (exp1){
    inst 1;
}else if(expe){
    inst 2;
}else if (expn){
    inst n
}else{
    inst n+1;
}inst n+2;
```

Switch: (sélectionner un traitement parmi plusieurs traitements)

```
switch(expression)
{case cond_1: traitement 1;
        [break;]
case cond_2 : traitement 2;
        [break;]
case cond_N : traitement N;
        [break;]
[default : <traitement N+1>;]
        [break;] }
```

ou bien:

```
if (condition)
    instruction;
else
    Instruction;
```

```
if(condition)
{ instruction 1;
  instruction 2;
  .....
}
```

```
else{
    instruction 2;
    .....
}
```


CamScanner

IV Structures répétitives :

1- la Boucle for : le nbre de répétitions peut être connu

Syntaxe : for (initialisation ; condition ; incrémentation) → ++ ou +4
{
Instruction à répéter ;
}

2- la Boucle while : le nbre de répétitions dépend d'une condition.

while : permet de répéter un bloc d'instruction tant qu'une condition est vrai.

Syntaxe : while (condition)
{
Instructions à répéter ;
}

3- la Boucle do...while : le nbre de répétitions dépend d'une cond.

do...while : permet de répéter un bloc d'inst. tant qu'une condition est vraie.

Syntaxe : do
{
instruction à répéter ;
} while (conditions) ;

4- la Boucle switch : (structure de contrôle)

Syntaxe :
```
switch (Expression) {
case val1 : Instruction 1;
            break ;
case valeur : Instr2;
...         break ;
case val N : Instr N ;
             break;
default : AutreInstruction ;
          Break ;
}
```

V - les fonctions d'entrées/Sorties :

Entrée non formatée

int getchar( ) ;
↳ lit et retourne un caractère provenant du clavier

Sortie non formatée

putchar (int c) ;
↳ affiche sur l'écran le contenu de "c". (le caractère qui correspond au code est mémorisé dans c)

a (97) – z (122) A (65) – Z (90)

Sortie formatée	Entrée formatée
Affichage formatée	Saisir formatée (depuis le clavier)
Printf ("format", [exp1,...expn]); printf ("expression");	scanf ("%d %d %d", &a, &b, &c);

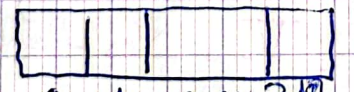
%d	Entier décimal	%c	caractère (char)	%p	Pointeur
%u	décimal non signé	%f	réel (float)		
%x	Entier hexadécimal	%s	chaine de caractère		

VI Les tableaux:

- il faut un type : tableau de caractère (seulement), tableau d'entier (seulement)....

Déclaration syntaxe : `Type nom_tab [Taille];`

Exp : `float N[3+8];`



Initialisation

oubien : `Type nom_Tab [Taille_N] = {val1, val2, ..., valN}`
`Type nom_Tab [ ] = {val1, val2, ..., valN}`

Acceder à un elt du tableaux :

syntaxe : `nom_tab [indice] = valeur;`

syntaxe Lecture : `scanf ("%...", &nom_tab[indice]);`
syntaxe écriture : `printf ("%...", &nom_tab[indice]);`

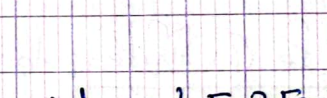
Remplir un tableau :

remplissage d'un tableau de N case

```
for (i=0 ; i<N ; i++){
    scanf("%d", &T[i]);
}
```

Matrice :

Initialisation : `int mat [2][3];`



`int mat [2][3] = { {1,2}, {3,4,5} };`

Initialisation par boucle :

```
for (int i=0 ; i<2 ; i++){
    for (int j=0 ; j<3 ; j++)
        mat [i][j] = i*j ;
}
```

```
{printf ("mat [%d][%d]", i, j);
 scanf("%d", &mat[i][j]);
}
```

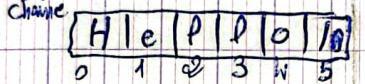
⋇ Chaines de Caractères (String)

Tableau de caractères

```
char chaine[6] = {'H','e','l','l','o','\0'};
```

Déclaration :

```
char chaine[] = "Hello"
```



Chaine de Caractère constante :

```
char *chaine = "Hello";
```

Affichage & écriture :

scanf ⟷ gets(ch) — lire d'une chaine (Lecture caractère par caractère du ch)

printf ⟷ puts(ch) — Affichage d'une Chaine (Affichage)

```
char chaine[10];
scanf("%s", chaine);
printf("%s", chaine);
```

Fonctions sur les Chaines dans (<string.h>)

```
#include <string.h>
....
```

```
char c[] = "Ali";
int n = strlen(c);   → 3
printf("%d", n);
```

```
char d[10];
strcpy(d,c);
puts(d)  → Ali
strncpy(d,c,2);
puts(d);  → Al
```

```
char d[10] = "Amine";
strcat(d,c);
puts(d);  → Amine Ali
strncat(d,c,2);  → Amine Al
                          (espace)
```

```
char a[] = "Ali";
char b[] = "Ali";
char c[] = "ali";
char d[] = "Amine";
printf("%d", strcmp(a,b));  → 0   Asci A = Asci A
printf("%d", strcmp(a,c));  → -1  Asci A < Asci a
printf("%d", strcmp(d,a));  → 1   Asci "m" > Asci A
```

0	2 ch identiques
négatif	valeur Asci 1er caract ch1 < val Asc ch2
positif	val Asci 1er caract ch1 > val Asci 1er caract ch2

```
char a[] = "Hello world"
printf("%s", strchr(a,'w'))  → World

char *b = strchr(a,'o');
printf("%s", b)  → o world
```

Recherche à l'ordre inverse (strrchr)

```
char a[] = "Hello world"
printf("%s", strrchr(a, 'l'));   → ld
char *b = strrchr(a, 'o');       → orld
printf("%s", b);
```

Expression arithmétique : (exp)

	A	B	C	D
A = 4+2;	6			
B = ++A;	7	7		
C = A*B++;	7	8	49	
D = pow(C,2);	7	8	49	2401
A/=2;	3.5	8	49	2401
D = ++D - C;	3.5	8	49	2353
C-- = B--;	3.5	7	41	2353
B = (A--+--C)/B;	2.5	6.21	40	2353

A diminue un 1 après calcul → ((A--) + (--C))/B
(3,5 ; 40) / 7 = 6,21

Exemples :

Boucle for

boucle qui affiche le message "Hello world" 5 fois

```
for (i=0; i<5; i++){
    printf("Hello world");
}
```

Boucle switch

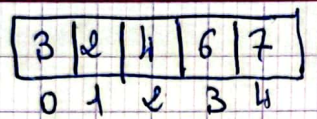
prog qui demandera à l'utilisateur de saisir un entier entre 1 et 7 au clavier, et afficher le nom du jour correspondant

N (1 ≤ ≤ 7) → programme
Si 16 → tenbre incon

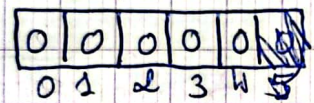
```
printf("veuillez saisir un entier entre 1 et 7:");
scanf("%d", &N);
switch(N){
    case 1 : printf("Lundi");
             break;
    ;
    case 7 : printf("Dimanche");
             break;
    default : printf("le nombre est incorrect");
              break;
```

Tableaux :

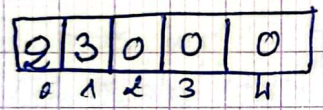
int T1[5] = {3,2,4,6,7};



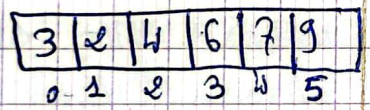
int T2[5] = { };



int T3[5] = {2,3};



int T4[ ] = {3,2,4,6,7,9}



int T5[5] = {1,2,3,4,9,8,5}

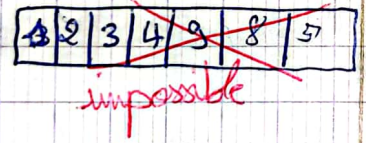


Tableau de caractères :

✓ char chaine[6] = {'H','e','L','L','o','\0'};

✗ char ch[4] = {'H','e','L','L','o','\0'};

✓ char ch[9] = {'H','e','L','L','o','\0'}; caractère \0 pour les autres cases

✓ char ch[ ] = {'H','e','L','L','o','\0'};

✓ char ch[6] = "Hello"; ajout à l'écriture case le caractère null '\0' automatiquement

✗ char ch[3] = "Hello";

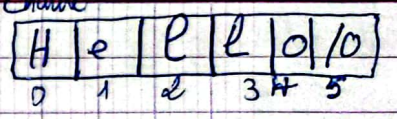
✓ char ch[8] = "Hello";

✓ char ch[ ] = "Hello"

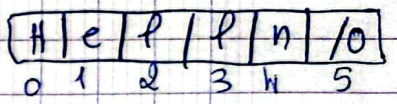
✗ char ch[6];
ch = "Hello";

EXP: (Affectation)

char chaine[ ] = "Hello"



chaine[4] = 'n'



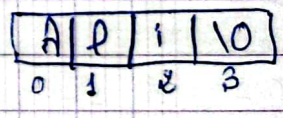
✗ Chaine de caractère cst:

✓ char *chaine = "Hello";

char *chaine;
chaine = "Hello";

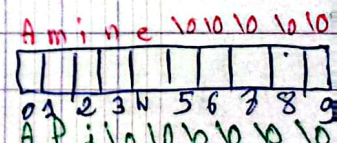
✗ Affichage et écriture:

✗ char c[4] = "Ali";
printf("%c", c[2]); → l



printf("%s", c); → Ali

✗ char c[10];
printf("Entrer votre nom :")
scanf("%s", c); Amine → Ali Amine



seulement Ali car scanf ne récupère pas l'espace

```
char C[10]
printf("Entrer votre nom");
gets(C);          → Ali Amine
printf("%s\n", C);   Ali Amine
printf("%c%c%c\n", C);   Ali
```

C	A	l	i		A	m	i	n	e	\0
	0	1	2	3	4	5	6	7	8	9

* Pas R & avec A (A)

```
void Puissance (int N){
    int P;
    P = N * N;
    Printf("La Puis de %d est : %d", N, P);
}
```

* Avec R & A (B)

```
int Puissance (int N) {
    int P;
    P = N * N;
    return P;
}
```

* (A)

```
int main(){
    int N;
    Printf("saisir la val de N:");
    scanf("%d", &N);
    Puissance (N);   ← appel de la fct
    return 0;
}
```

(B)

```
int main(){
    int N, P;
    Printf("saisir ___ N:");
    scanf("%d", &N);
    Printf("La Puis de %d est %d", N, P)
    return 0;
}
```

déclaration de V1

```
struct Personne {
    char nom[12];
    int age;
    double taille;
};

void main(){
    int P;
// appel de new type V1
struct Personne Pers;
```

déclaration de V2

```
typedef struct {
    char nom[12];
    int age;
    double taille;
} Personne;

void main(){
// appel de mon type V2
Personne Pers;
```

```
typedef struct datn{
    int jour;
    int mois;
    int annee;
} datn;
typedef struct {
    int nom[12];
    int age;
    double taille;
// datn date;
typedef struct {
    int jour;
    int mois;
    int annee;
} datn;
} Personne;
void main(){
Personne P1, P2;
Printf("Donner le jour de la
    date de naissance de P1");
scanf("%d", &P1.date.jour);
```

oubliez

```
typedef struct {
    char nom[15];
    int age;
    int tel;
} Personne;
```

```
void main(){ personne P1, P2;
Printf("information du Pers num1:\n");
scanf("%s", &P1.nom);
printf("_____ num2\n");
scanf("%s", &P2.nom);
printf("taper l'age du pers num 1")
scanf("%d", &P1.age);
```

Autre exp

```
struct livre
{
    char nom_auteur[15];
    char titre[20];
    int an;
} l;
l.an = 1993;
strcpy(l.nom_auteur, "Molaire");
struct livre l = {"Dupont", 'c', 2012}; ← Initialisation
```

```
typedef struct AdresseClient{
    char rue[20];
    int code;
    char ville[20];
} adcl;
adcl ≡ AdresseClient
adcl a; ⟺ struct AdresseClient a;
```

```
//Affichage
Printf("%s , %d , %d \n", P1.nom, P1.age, P1.tel);
```

Déclaration : Type nom_tab [N] [M] = { { ... }, { ... } ... }

```
int mat[3][4];
int mat[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

1	2	3	4
5	6	7	8
9	10	11	12

Initialisation

```
for (i=0; i<3; i++){
    for (j=0, j<4, j++){
        printf("mat[%d][%d]", i, j);
        scanf("%d", &mat[i][j]);
    }
}
```

Accéder à un élt :

```
Printf("%...", nom_tab[num_lig][num_lig]
```

Syntaxe du tableau de 2 dimension :

affectation : nom_tab[val][num] = val;

Lecture : scanf("%...", &nom_tab[num][num]);

Ecriture : printf("%...", nom_tab[num][num]);

Déclaration de la fonction :

arguments

```
type_retour nom_fcts ( type arg1, type arg2, ...){
        inst 1;
        inst 2;
        ...;
        return resultat;
}
```

* Pas retour & pas d'arguments

```
void nom_fcts(){
    Inst;
}
```
(A)

* Avec retour & pas d'argument

```
type_retour nom_fcts(){
    Inst;
    return resultat;
}
```
(C)

* Pas retour & avec arguments

```
Void nom_fcts (type args,..){
    Inst;
}
```
(B)

* Avec retour & avec Arguments

```
type_retour nomfcts(type args,..){
    Inst;
    return resultat;
}
```
(D)

Affectation

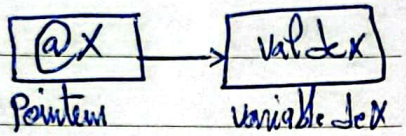
(A) . nom_fun();

(B) . nom_fun (arg1, arg2, ...)

(C) . X = nom_fun();
ou . X = Y / nom_fun();
ou . printf (nom_fun());

(D) . X = nom_fun (arg1, arg2, ...
. X = Y / nom_fun (arg1, arg2, ...
. printf (nom_fun (arg1, arg2, ...

Les Pointeurs:



"=" affectation

① . variables et adresses

int x;
&x ← adresse de x dans la mémoire en Hexadécimal
int tab[10];
tab ← adresse du premier élt du tab (&tab[0])

Syntaxe: (variables pointeurs)

<type-données> * <nom-pointeur>

Exp: int *pi; ← la variable pi est un pointeur vers une donnée de type int
char *pc; ← pc est un pointeur de type char vers un

Initiation: int x; // variable d'entier
int *pi; // variable pointeur
float *pf;
float y;

pi = &x; pi est un pointeur vers int, initialisé avec l'adresse de la variable x.

Affichage: Printf("%x", &x); Hexadécimal
ou "%d" décimal

T: l'adresse de T[0]	*(T+i): le contenu de T[i]	p+i: pointe sur T[i]
T+i: l'adresse de T[i]	P: pointe sur T[0]	*(p+i): le contenu de T[i]

int T[N]; int *P; P=T

② Accès indirect aux variables: on utilise "*" opérateur d'indirection

Pour accéder à une autre donnée

Exp: x = 3;
pi = &x;
y = *pi; (⇒ y = x)

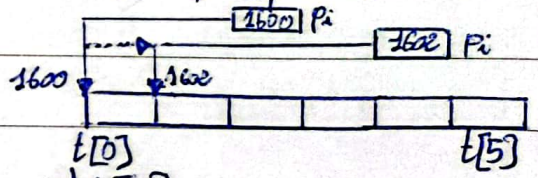
③ Les opérateurs définis sur les Pointeurs:

• Addition:

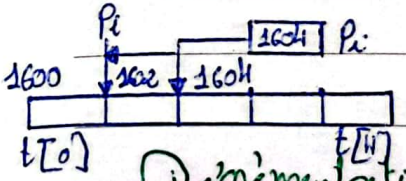
donc
int t[6];
int *pi;
pi = &t[0]; // pi pointe sur le 1er elt du tableau 't' vers la variable "int" t[0]

l'instruction:

pe = pi + 1; // l'adresse dans pi est passé de 1600 à 160



Incrémentation: int i[10];
pi = &i[0]; avec int *pi;



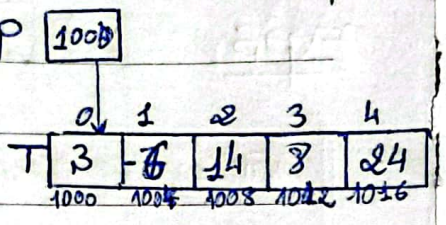
pi = pi + 1; (⇒ pi++; ou ++pi)
(*pi)++; // on augmente de 1 la valeur de la donnée pointée actuellement par pi

Décrémentation:

pi = pi - 1; (⇒ pi--; ou --pi)

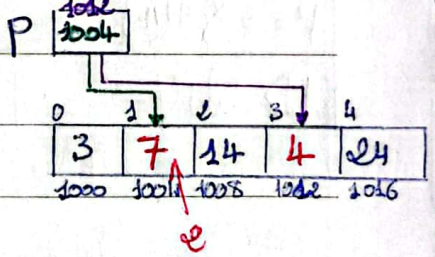
Exp1 :

Pointeur – Tableau



int T[5] = {3, -6, 14, 8, 24};

int *p;	T : l'adresse de T[0]
p = T;	T+i : l'adresse de T[i]
*(T+1) = 2; # T[0+1] = 2	*(T+i) : le contenu de T[i]
p++; ← 1004	P : Pointe sur T[0]
*(T+3) = 4; T[0+3] = 4	P+i : Pointe sur T[i]
*p+4 = 5; p = p+5 = 2+5 = 7	*(P+i) : le contenu de T[i]
p = p+2; ← T[2] 1012	



Le nom d'un tableau représente l'adresse de son 1er élt.

T : l'adresse de l'élt T[0]	P+i : Pointe sur l'élt T[i]
T+i : l'adresse de l'élt T[i]	*(P+i) : Le contenu de l'élt T[i]
*(T+i) : le contenu de l'élt T[i]	
Si P = T	
P : Pointe sur l'élt T[0]	

Exp2 :

Instruction	A	B	C	P1	P2
Initialisation	1	2	3	-	-
P1 = &A;	1	2	3	&A	-
P2 = &C;	1	2	3	&A	&C
*P1 = (*P2)++;	3	2	4	&A	&C
P1 = P2;	3	2	4	&C	&C
P2 = &B;	3	2	4	&C	&B
*P1 -= *P2	3	2	2	&C	&B
++*P2;	3	3	2	&C	&B
*P1 *= *P2;	3	3	6	&C	&B
A = ++*P2 * *P1;	24	4	6	&C	&B
P1 = &A;	24	4	6	&A	&B
*P2 = *P1 / = *P2	6	6	6	&A	&B

int T[5] = {3, -6, 14, 8, 24};
int *p;
p = T;
printf("%d", sizeof(T[0])); → 4 car 4 octets dans la mémoire
printf("%d", sizeof(double)); → 8

sizeof() taille (en octets) du type ou de variable qui suit.

Incrémentation

p++; ↔ &T[0+1];

* Pour le dernier exp:

```
p++;
printf("%p", *p);  ↓ 7
printf("%d", *(p++));  ↓ 7
```

Décrémentation

p--; ↔ &T[2-1];

* Pour le dernier exp:

```
p--;
printf("%d", *p);  ↓ 7
```

Soustraction:

q-2; ↔ &T[3-2];

q-2; ↔ q = q - 2*sizeof(int) ↔ p = 1012 - 2×4 = 1004

```
int *q;
q = &T[3];
printf("%p", q-2);  ↓ 1004
printf("%d", *(q-2));  ↓ 7
```

Comparaison:

```
printf("%d", q==p);  ↓ 0
```

Addition

P = P+3; ↔ &T[0+3];

P = P+3 ↔ p = p+3*sizeof(int) ↔ p = 1000 + 3×4 = 1012

```
printf("%p", p+3);  ↓ 1012
printf("%d", *(p+3));  ↓ 4
```

q-p; ↔ Distance entre P et q

q-p ↔ (q-p)/sizeof(int) ↔ (1012-1000)/4 ↔ 3

```
printf("%d", q-p);  ↓ 3
printf("%d", p-q);  ↓ -3
```

EXP3:

	0	1	2	3	4	5	6	7	8
A	12	23	34	45	56	67	78	89	90

P: &23 → &12

Instruction	Résultats
++P;	&23
*(p+3);	56
P - &A[4];	-3
--P;	&12
A+6;	&78
(P+(P+8)-A[7]);	23
P == &A[4];	0
P >= A;	1

CamScanner

Tableaux des entiers

T = (int *) malloc (N sizeof(int));

T+0 ⇔ &T[0]	T ⇔ T[0]
T+1 ⇔ &T[1]	T+1 ⇔ T[1]
T+N-1 ⇔ &T[N-1]	Point *p;
	(*p).x ⇔ p->x
	(*p).y ⇔ p->y
	(*p).c ⇔ p->c