L'ESSENTIEL du MODULE INFO1 :

PROGRAMMATION en C

Sommaire

1) L'ESSENTIEL des BASIQUES

Manipuler des variables et des constantes,
Ecrire des instructions de lecture clavier et d'écriture écran,
Faire des calculs arithmétiques avec des entiers et des réels (addition, multiplication, divisions, modulo...),
Savoir utiliser et remplir des fonctions simples.

1.1) Manipuler des données (VARIABLES et CONSTANTES)

➢ *TYPES DES DONNEES EN LANGAGE C :*

TYPE	Signification	CAPACITE	TAILLE (en octets)
char / unsigned char	Caractère	[-128,127] / [0,255]	1
unsigned short	Entier non signé court	[0,65 535]	2
short	Entier signé court	[-32 768,32 767]	2
unsigned long ou unsigned int	Entier non signé long	[0,4 294 967 295]	4
long ou int	Entier signé long	[-2147483648,2147483647]	4
float	Réel simple précision	[+- 3.4 10^{38}, +- 3.4 10^{38}]	4
double	Réel double précision	[+-1.7 10^{308}, +-1.7 10^{308}]	8

➢ *INSTRUCTION POUR DECLARER UNE VARIABLE NUMERIQUE (réel, entier) :*

INSTRUCTION : ***TYPE nomVariable;*** // instruction de **DECLARATION**

Qualité : nom explicite, pas long, commençant par une minuscule

EXEMPLES :

- **double moy;** **// réel de nom « moy »**
- **int compteur;** **// entier de nom « compteur »**

➢ *INSTRUCTION POUR INITIALISER UNE VARIABLE NUMERIQUE (réel, entier) :*

INSTRUCTION : **nomVariable= valeur;** // instruction d'**AFFECTATION**

Dans partie données ou instructions

EXEMPLE :

- **moy= 15.5;** **// la moyenne est de 15.5**
- **compteur= 0;** **// le compteur démarre à 0**

➢ *INSTRUCTION POUR DECLARER UNE CONSTANTE :*

INSTRUCTION : ***const TYPE NOMCONSTANTE= valeur ;***

Qualité : nom explicite, court, en MAJUSCULES

EXEMPLES :

- **const double PI= 3.14;**
- **const int NB_NOTES= 30;**

1.2) Ecrire sur l'écran : PRINTF()

➢ *PRINCIPAUX FORMATS DU PRINTF() :*

Format	Signification		Format	Signification
%c	char			
%hu	unsigned short		**%hx**	unsigned short affiché en Hexadécimal
%hd	short		**%ld ou %d**	long ou int
%lu	unsigned long ou unsigned int		**%lx**	unsigned long affiché en Hexadécimal
%f	float		**%.4f**	float avec, au maximum, 4 décimales
%lf	double		**%.2lf**	double avec, au maximum, 2 décimales

➢ *INSTRUCTION POUR ECRIRE DU TEXTE SUR L'ECRAN DE L'ORDINATEUR :*

INSTRUCTION : **printf("\tTexte\n");**

Inclure bibliothèque stdio.h

EXEMPLE : **printf("\n\tBonjour !\n"); // affiche "Bonjour", après un passage à la ligne et une tabulation ; puis, passe en début de ligne suivante**

➢ *INSTRUCTION POUR ECRIRE DES VALEURS DE VARIABLES SUR L'ECRAN :*

INSTRUCTION : **printf("Texte : %format",nomVariable);**
avec plusieurs variables : **printf("a : %format\tb : %format", a, b);**

Inclure bibliothèque stdio.h

EXEMPLES :

- **printf("Moyenne : %.2lf\n",moy); // réel moy affiché avec 2 décimales**
- **printf("a:%d\nb :%hu\nc :%lx",a,b,c); // affichage de 3 entiers, dont le dernier en hexadécimal**

1.3) Saisir des valeurs au clavier : SCANF()

➢ *PRINCIPAUX FORMATS DU SCANF() :*

Format	Signification	Format	Signification
%c	char		
%hu	unsigned short	**%hd**	short
%lu	unsigned long ou unsigned int	**%ld ou %d**	long ou int
%f	float	**%lf**	double

➢ *INSTRUCTION POUR SAISIR DES VALEURS DE VARIABLES AU CLAVIER :*

INSTRUCTION : **scanf("%format",&nomVariable);**
avec plusieurs variables : **scanf("%format%format ", &a, &b);**

Inclure bibliothèque stdio.h

EXEMPLES :

- **scanf("%lf",&moy); // saisit une valeur au clavier et la stocke dans le double « moy »**
- **scanf("%d%hu%hd",&a,&b,&c); // saisie de 3 valeurs au clavier et stockage dans 3 entiers. L'utilisateur du programme sépare les 3 nombres, sur le clavier, par : ESPACE, TAB ou ENTER**

1.4) Ecrire un calcul

➢ *EXPRESSION ARITHMETIQUE :*

DEFINITION **: expression avec des opérateurs arithmétiques pour les calculs mathématiques.**
Opérateurs arithmétiques : addition +, soustraction -, multiplication *, division /, modulo %

EXEMPLES **:**

- **(a*b+3.) / (c-5.6)** **// division réelle**
- **a%b** **// reste division entière de a par b (entiers)**

➢ *INSTRUCTION de CALCUL :*

INSTRUCTION **: *nomVar= formuleCalcul;*** // instruction de **CALCUL**
FormuleCalcul : expression arithmétique AVEC/SANS fonction mathématique de <math.h>

PRECISIONS **:** Règles d'évaluation d'une formule de calcul :
1. L'ordre de calcul dépend des **parenthèses** (forcent les priorités).
2. L'ordre de calcul dépend, ensuite, de la **priorité des opérateurs** (voir Tableau PRIORITES).
3. Pour un même niveau de priorité, l'évaluation de l'expression se fait **de gauche à droite** (à partir du =).

EXEMPLES :

- **moy= (a+b)/2.;** **// calcul moyenne de 2 réels**
- **aire= PI*pow(r,2);** **// aire disque de rayon r**

➢ *DIVISION ENTIERE et REELLE :*

DEFINITION **: nomVar= a/b**
La division produit un résultat entier ou réel selon les types des données.
1. DIVISION ENTIERE : SI a et b sont entiers, l'ALU fait une division entière (résultat de a/b est entier).
2. DIVISION REELLE : SI a ou b est réel, l'ALU fait une division réelle (résultat de a/b est réel).
3. SI nomVar est ENTIER : quelque soient a et b, le résultat de a/b est tronqué en entier lors du stockage dans la variable nomVar.

PRECISIONS **: Pour éviter les erreurs de calculs, utiliser, au maximum, les mêmes types dans un calcul.**

EXEMPLES :

- **5/2 vaut 2 et 5%2 vaut 1** **// division entière**
- **5/2. vaut 2.5** **// division réelle**
- **a= 5/2.** **// si a est un int, division réelle tronquée : a vaut 2**

1.5) Les opérateurs de calcul

➢ *OPERATEURS ARITHMETIQUES, BINAIRES et LOGIQUES :*

Opérateurs Arithmétiques

Opérateur	Nom	En C
-	Soustraction	-
X	Multiplication	*
/	Division	/
%	modulo	%

Opérateurs Binaires

Opérateur	Nom	En C
+	Ou binaire	|
.	Et binaire	&
ā	Complément à 1 (inverseur)	~a
⊕	Ou exclusif	^

Opérateurs Logiques de Comparaison

Opérateur	Nom	En C
==	Egal	==
≠	Différent	!=
>	Strictement supérieur	>
≥	Supérieur ou égal	>=
<	Strictement inférieur	<
≤	Inférieur ou égal	<=

Opérateurs Logiques Booléens

Opérateur	Nom	En C
NON	Non logique	!
OU	Ou logique	||
ET	Et logique	&&

➢ *LES PRIORITES des OPERATEURS POUR les CALCULS :*

Priorité	Opérateur
1	()
2	NON (*non logique*) — (*complément à 1*)
3	x (*multiplication*) / (*division*) % (*modulo*)
4	+ (*addition*) - (*soustraction*)
5	< > ≤ ≥
6	== (*égalité*) ≠ (*différent*)
7	. (*et binaire*)
8	⊕ (*ou exclusif*)
9	+ (*ou binaire*)
10	ET (*et logique*)
11	OU (*ou logique*)
12	= (*affectation*)

29

➢ *DIFFERENCE ENTRE LES OPERATEURS = (affectation) et == (égalité) :*

DEFINITION : **= est l'opérateur d'affectation qui attribue (affecte) une valeur à une donnée.**
DEFINITION : **== est l'opérateur d'égalité qui permet de comparer deux valeurs.**

EXEMPLES :

- **a= b;** // a reçoit la valeur de b
- **c= 5;** // c reçoit la valeur 5
- **a==b;** // valeurs de a et b comparées
- **c==5;** // si c vaut 5, l'expression est true

1.6) Commencer à utiliser des fonctions

➢ *UTILISER DES FONCTIONS STANDARDS :*

DEFINITION **: Une fonction est un bloc d'instructions réutilisable.**

```
Prototype de la fonction ou #include <fichier Header standard>
int main()
{  …     Appel de la fonction standard ;  … }
```

PRECISIONS **:**

1. PROTOTYPE : déclaration de fonction (mode d'emploi de la fonction). Le prototype indique ce qu'il faut donner à la fonction pour son traitement et ce que la fonction renvoie comme résultat ; ex. **double** COS**(double** *alpha***);**.
2. APPEL : instruction lançant l'exécution d'une fonction, elle doit respecter le prototype ; ex. cosinus= cos(alpha).

EXEMPLES :

```
#include <math.h>
int main()
{   double   alpha, cosinus;
    printf("\tDonnez angle : ");  scanf("%lf",&alpha);  // saisie angle
    cosinus= cos(alpha);   // appel fonction
    printf("\n\tCosinus: %.2lf\n",cosinus);                }
```

➢ *METTRE EN PLACE DES FONCTIONS SIMPLES :*

1. PROTOTYPE fonction en haut du fichier source (ou dans un Header) :

```
TYPESortie  NomFonction(TYPEEntrées) ;          // PROTOTYPE fonction = mode d'emploi
int main() {    }
```

2. Définition de la fonction (ses instructions), souvent sous le main() :

```
int main() {    }
TYPESortie  NomFonction(TYPEEntrées)          // DEFINITION fonction = description
{       données et instructions ;          }
```

3. Appeler la fonction depuis le main() :

```
int main()
{   déclaration verResu et varEntrées ;
    varResu=NomFonction(varEntrées) ;          // APPEL fonction = exécution          }
```

EXEMPLE :

// Fichier facture.cpp

```
float CalculerPrixTTC(float pHT);   // prototype fonction
int main()
{    float prixHT, prixTTC;
     printf("Donner HT : ");     scanf("%f",&prixHT);      // saisie prixHT
     prixTTC=CalculerPrixTTC(prixHT);    // appel de la fonction de calcul du TTC
     printf("\nPrix TTC : %.2f",prixTTC);   // affichage de la facture
}
float CalculerPrixTTC(float pHT)        // définition fonction
{    float pTTC;                  const float TVA= 0.196;
     pTTC= pHT * (1+TVA);
     return(pTTC);
}
```

PRECISIONS **:** RETURN à la fin de la fonction : pour sortir de la fonction et renvoyer, éventuellement, un résultat.

2) L'ESSENTIEL des STRUCTURES de CONTROLE

Instructions alternatives if{} else{} et if{} else if{} else{}, switch{} et Instructions itératives for(){},do{} while(), while{}.

2.1) Alternatives

➢ *EXPRESSION LOGIQUE :*

DEFINITION **: une expression logique utilise des opérateurs logiques. Sa valeur est true (VRAI-valeur différente de 0) ou false (FAUX-valeur 0) ; elle sert aux conditions de certaines instructions.**

PRECISIONS **:** voir les opérateurs logiques dans la partie Calculs.

EXEMPLES **: si a vaut 3, b vaut 10 et c vaut 5 :**

- **(a !=b) && (a==c)** **// (a≠b)ET(a==c) ⇒false**
- **(c<b)|| !a** **//(c<b)OU NON(a) ⇒true**

➢ *INSTRUCTION ALTERNATIVE SI ou SI-SINON :*

INSTRUCTION **:**

```
if (condition logique)
{       instructions 1;
}
else    // optionnel
{ instructions 2;
}
```

PRECISIONS **:** le SINON (else) est OPTIONNEL

EXEMPLES :

```
// calcul de prix sans ou avec remise
prixTot= nbProd *PRIX_UNIT;
if (nbProd>=10)
{       prixTot= prixTot * 0.8;
}
```

```
// calcul de prix sans ou avec remise
if (nbProd<10)  // prix sans réduction
{       prixTot= nbProd *PRIX_UNIT;
}
else   // 20% si au moins 10 articles achetés
{       prixTot= nbProd*PRIX_UNIT*0.8;
}
```

➢ *INSTRUCTION ALTERNATIVE SI-SINON SI :*

INSTRUCTION : ***if (condition logique 1)***
{ instructions 1;
}
else if (condition logique 2)
{ instructions 2;
} …
else // optionnel
{ instructions 3;
}

PRECISIONS :
- le SINON (else) est OPTIONNEL
- le nombre de SINON SI (else if) n'est pas limité

EXEMPLE :
```
// prix avec remises progressives
if (nbProd<10) // [1,9] articles : pas réduction
{       prixTot= nbProd *PRIX_UNIT;
}
else if (nbProd<100)  // [10,99] articles : 20%
{       prixTot= nbProd*PRIX_UNIT*0.8;
}
else // au moins 100 articles : 50%
{       prixTot= nbProd *PRIX_UNIT*0.5;
}
```

➢ *INSTRUCTION ALTERNATIVE CHOIX-CAS :*

INSTRUCTION : ***switch(nomVar) // entier ou caractère***
{ case val1 : instructions 1; break ;
case val1 : instructions 2; break ;
…
default : instructions 3; // optionnel
}

PRECISIONS :
- le DEFAUT (default) est OPTIONNEL
- le nombre de CAS (case) n'est pas limité
- nomVar est une variable, forcément de type ENTIER ou CARACTERE (pas réel)

EXEMPLE :
```
// gestion menu du programme
switch(choixMenu)
{   case 1:   resu= a+b ;          // Addition
              break;
    case 2:   resu= a-b ;          // Soustraction
 }
```

2.2) Boucles

➢ *INSTRUCTION ITERATIVE POUR :*

INSTRUCTION :	*PRECISIONS* :
Comptage (pas>0, valInit≤ValFin) : ***for (i=valInit ; i<=valFin ; i=i+pas)*** ***{ instructions;*** ***}*** **Décomptage (pas<0, valInit≥ValFin) :** ***for (i=valInit ; i>=valFin ; i=i-pas)*** ***{ instructions;*** ***}***	1. Initialisation : ***i=valInit*** ; i, variable compteur, doit être de type entier 2. Condition logique de poursuite de boucle : ***i<=valFin*** (ou i>=valFin pour le décomptage). 3. Condition logique de fin de boucle : ***i>valFin*** (ou i<valFin pour le décomptage). 4. Incrémentation : ***i=i+pas*** ; pas : incrément entre 2 exécutions du POUR.

EXEMPLE :

```
// somme cumulée de saisies clavier
int     a, somme, i ;

somme= 0;   // initialisation de la somme
for(i=1 ;i<=4 ;i=i+1)          // comptage de 4 saisies
{     printf("Donner un entier : ") ;
      scanf("%d",&a) ;
      // à chaque passage: cumul de la nouvelle saisie a avec la valeur précédente de somme
      somme= somme + a;
}
printf("somme : %d",somme);              // affichage de la somme des 4 entiers
```

➢ *INSTRUCTION ITERATIVE FAIRE – TANT QUE :*

INSTRUCTION : ***do***
{ instructions;
} while (condition);

EXEMPLE :

```
// saisie validée d'un entier
int     x;

do   // reprise de saisie si saisie invalide
 {      printf("Donner entier dans [5,150] : ");
        scanf("%d",&x);
        if (x<5 || x>150)  // message saisie invalide
              printf("Erreur de saisie") ;
  } while (x<5 || x>150 );
```

➢ *INSTRUCTION ITERATIVE TANT QUE :*

INSTRUCTION : ***while (condition)***
{ instructions;
}

EXEMPLE :

```
// table de multiplication par 2
int  prod, i;

i= 0;   // initialisation du comptage
while (i<11) // parcours des valeurs [0,10]
{       prod=i*2; printf("%dx2=%d\n",i,prod);
        i++;      // passage à l'entier suivant
}
```

3) L'ESSENTIEL des NOTIONS AVANCEES

Manipuler des données Structurés ou de type Tableau, créer des fonctions : sans échanger de données, avec des paramètres en Entrée, avec un résultat en Sortie ou avec des paramètres en Entrée/Sortie.

3.1) Structures

➢ *CREER UN TYPE STRUCTURE :*

INSTRUCTION : une structure est composée de différents champs (de types simples quelconques) caractérisant une entité :

```
struct   NomStructure
{   TYPEchamps1     nomChamps1;
    TYPEchamps2     nomChamps2;
    ...
};
```

EXEMPLE :

```
// structure décrivant une personne
struct   structPersonne
{   char     initialNom;
    char     initialPrenom;
    int      age;
    double   salaire;
};
```

➢ *DECLARER UNE VARIABLE DE TYPE STRUCTURE :*

INSTRUCTION : `struct NomTYPEStructure nomVariable;`

EXEMPLE :

```
// déclaration d'une variable du type structuré « structPersonne »
struct   structPersonne  florian;
```

➢ *INITIALISER UNE VARIABLE STRUCTURÉE :*

INSTRUCTION : `nomVariable.nomChamps= valeur;`

EXEMPLE :

```
// affectation de valeurs aux champs de la variable structurée
florian.initialNom= 'T';
florian.initialPrenom= 'F';
florian.age= 25;
florian.salaire= 1900.;
```

3.2) Tableaux

➢ *DECLARER UN TABLEAU :*

INSTRUCTION : ***TYPECasenomTableau[NBCASE]; // Recommandé: NBCASE= constante***

EXEMPLE : **double tabNotes[50]; // tableau de 50 réels double**

➢ *INITIALISER UN TABLEAU DANS LA DECLARATION :*

INSTRUCTION : ***TYPECase nomTableau[NBCASE]= {val1,.,valn};***

EXEMPLE : **int tabMesures[3]= {-2, 6, 100}; // tableau de 3 entiers**

➢ *INITIALISER UN TABLEAU PAR AFFECTATIONS :*

INSTRUCTION :
```
for(i=0 ;i<NBCASE;i++)
{  nomTableau[i]= valeur;  }
```

EXEMPLE : **for (i=0 ; i<3 ; i++) { tabMesures[i]= 0; } // initialisation à 0 de toutes les cases**

➢ *INITIALISER UN TABLEAU PAR TIRAGES ALEATOIRES :*

INSTRUCTION :
```
srand(time(NULL));  // au début du main(), à faire une seule fois dans le programme
for(i=0 ;i<NBCASE;i++)
{  nomTableau[i]= rand() % (valMax+1);  // pour tirer une valeur dans [0,valMax]
}
```

REMARQUE : inclure stdlib.h et time.h

➢ *INITIALISER UN TABLEAU PAR SAISIES CLAVIER :*

INSTRUCTION :
```
for(i=0 ;i<NBCASE;i++)
{  printf("donner la case %d : ",i);
   scanf("%format",&nomTableau[i]);
}
```

EXEMPLE :
```
for (i=0 ; i<3 ; i++)  {  printf ("\nCase d'indice  %d : ",i);
                          scanf("%d",&tabMesures[i]);  }
```

➢ *AFFICHER LE CONTENU D'UN TABLEAU :*

INSTRUCTION :
```
for(i=0 ;i<NBCASE;i++)
{  printf("%format  ",nomTableau[i]);  }
```

EXEMPLE : **for (i=0 ; i<3 ; i++) { printf ("\nCase num%d:%d",i,tabMesures[i]); }**

➢ *CUMUL DES VALEURS D'UN TABLEAU :*

INSTRUCTION :
```
cumul=valeurNeutre;
for(i=0 ;i<NBCASE;i++)
{  cumul=cumul  + ou *  nomTableau[i];  }
```

EXEMPLE :
```
int tabMes[3]={1,2,5},i, somCumul=0;
/* somme cumulée */  for (i=0 ; i<3 ; i++)  {  somCumul= somCumul+tabMes[i] ;  }
                     printf ("\nSomme mesures: %d",somCumul);
```

3.3) Fonctions

➢ *FONCTION SANS ECHANGE DE DONNEES :*

DEFINITION **:**

1. PROTOTYPE fonction (sa DECLARATION), en haut du fichier source :

```
void  NomFonction(void) ;          // prototype fonction
int main() {   }
```

2. DEFINITION fonction (ses instructions) :

```
void  NomFonction(void) ;          // prototype fonction
int main() {   }
void  NomFonction(void)            // définition fonction
{       données et instructions ;          }
```

3. APPEL fonction depuis le main() :

```
void  NomFonction(void) ;          // prototype fonction
int main()
{   NomFonction() ;                // appel fonction
}
void  NomFonction(void)            // définition fonction
{       données et instructions ;     }
```

EXEMPLE :

// Fichier aide.cpp

```
// PROTOTYPE - Affichage fenêtre d'aide
void AfficherAide(void);
_____________________________________________________
int main()
{     // APPEL - Affichage fenêtre d'aide
      AfficherAide();
}
_____________________________________________________
// DEFINITION - Affichage fenêtre d'aide
void AfficherAide(void)
{     printf("\nAide Mastermind\n\t--------\n");
      printf("Blablabla…");
}
```

➢ *FONCTION AVEC PARAMETRES EN ENTREE (E) :*

DEFINITION **:**

4. PROTOTYPE fonction (sa DECLARATION), en haut du fichier source :

```
void NomFonction(TYPEparamètreE) ;                    // prototype fonction
int main() {    }
```

5. DEFINITION fonction (ses instructions) :

```
void NomFonction(TYPEparamètreE) ;                    // prototype fonction
int main() {    }
void  NomFonction(TYPEparamètreE   Nomparamètre)              // définition fonction
{       données et instructions ;          }
```

6. APPEL fonction depuis le main() :

```
void NomFonction(TYPEparamètreE) ;                    // prototype fonction
int main()
{   Déclarer NomDonnée
    NomFonction(NomDonnée) ;                          // appel fonction
}
void  NomFonction(TYPEparamE   Nomparamètre)                  // définition fonction
{       données et instructions ;      }
```

EXEMPLE :

// Fichier facture.cpp

```
// PROTOTYPE - Affichage facture
void AfficherFacture(float);
_____________________________________________________________________
int main()
{   float  prixHT;

    // Saisie du prix Hors Taxe de l'article
    printf("Donner prix HT : ");
    scanf("%f",&prixHT);
   // APPEL - Affichage facture avec prix TTC
    AfficherFacture(prixHT);
}
_____________________________________________________________________
/* DEFINITION - Affichage facture      - Entrée : prix Hors Taxe du produit */
void AfficherFacture(float pHT)
{   const   float   TVA= 0.19 ;

    printf("\n\tFacture\n\tPrix HT\t\tPrix TTC\n\t%.2f\t%.2f", pHT, pHT*(1+TVA));
}
```

PRECISION **:** Un PARAMETRE en ENTREE est **passé par valeur**. La donnée transmise (ex. prixHT) est en lecture seulement pour la fonction appelée, qui ne peut pas la modifier. La fonction reçoit une valeur, mais ne peut pas accéder à la zone RAM de prixHT.

➢ *FONCTION AVEC RESULTAT EN SORTIE (S) :*

DEFINITION :

1. PROTOTYPE fonction (sa DECLARATION), en haut du fichier source :

```
TYPESORTIE  NomFonction(void) ;                    // prototype fonction
int main() {    }
```

2. DEFINITION fonction (ses instructions) :

```
TYPESORTIE  NomFonction(void) ;                    // prototype fonction
int main() {    }
TYPESORTIE  NomFonction(void)                      // définition fonction
{       TYPE SORTIE     variableSortie;
        return(variableSortie) ;                }
```

3. APPEL fonction depuis le main() :

```
TYPESORTIE  NomFonction(void) ;                    // prototype fonction
int main()
{    TYPE SORTIE     nomVariable;
     nomVariable= NomFonction() ;               }     // appel fonction
TYPESORTIE  NomFonction(void)                      // définition fonction
{       TYPE SORTIE     variableSortie;
        return(variableSortie) ;                }
```

EXEMPLES :

// Fichier factoriel.cpp

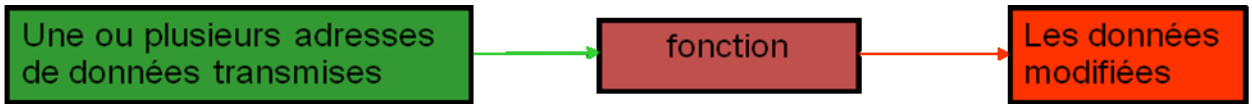
```
// PROTOTYPE - Calcul du factoriel de 7
int CalculerFacto(void);
_____________________________________________________________________
int main()
{    int factoriel ;

     factoriel= CalculerFacto();          // APPEL - Calcul et affichage de 7!
     printf("\nFactoriel : %d",factoriel);
}
_____________________________________________________________________
/* DEFINITION - Calcul du factoriel de 7.
- Sortie : 7 ! */
int CalculerFacto (void)
{    int  facto= 1, i;

     /* calcul de : 1 x 2 x 3 x ... x 7*/
     for ( i=2 ; i<=7 ; i=i+1 )     {     facto= facto*i;     }
     return(facto);
}
```

PRECISION : Une fonction, en langage C, ne peut renvoyer qu'une seule SORTIE.

➢ *FONCTION avec PARAMETRE en E/S de TYPE TABLEAU :*



DEFINITION **:**

1. PROTOTYPE fonction (sa DECLARATION), en haut du fichier source :

```
void  NomFonction(TYPETableau nomTab[]) ;                    // prototype fonction
int main() {    }
```

2. DEFINITION fonction (ses instructions) :

```
void  NomFonction(TYPETableau nomTab[]) ;                    // prototype fonction
int main() {    }
void  NomFonction(TYPETableau nomTab[])                      // définition fonction
{       données et instructions ;
}
```

3. APPEL fonction depuis le main() :

```
void  NomFonction(TYPETableau nomTab[]) ;                    // prototype fonction
int main()
{    TYPETableau   nomTableau[NBCASE];
     NomFonction(nomTableau) ;                               // appel fonction
}
void  NomFonction(TYPETableau nomTab[])                      // définition fonction
{       données et instructions ;                    }
```

EXEMPLE **:**

// Fichier tableau.cpp

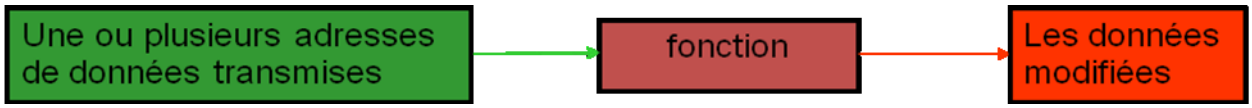
```
void RemplirTableau(float  tabR[2]); // PROTOTYPE - Initialisation d'un tableau
_____________________________________________________________________________
int main()
{    float  tabReel[2];   int i;

     RemplirTableau(tabReel);              // APPEL – Initialisation du tableau
      for(i=0;i<2;i++)                            // Affichage du tableau de 2 réels
     {    printf("\t%.2f",tabReel[i]);          }
}
_____________________________________________________________________________
/* DEFINITION - Initialisation d'un tableau.  - Paramètre modifié (E/S) : le tableau tabR */
void RemplirTableau(float  tabR[2])
{     int i;

      for ( i=0 ; i<=1 ; i=i+1 )     {       tabR[i]= i+1;              }
}
```

PRECISION **:** Un PARAMETRE en ENTREE/SORTIE est **passé par adresse**. La donnée transmise (ex. tabReel) est en lecture/écriture pour la fonction appelée, qui peut la modifier. La fonction a l'accès direct à la zone RAM de tabReel. La modification de tabR entraîne la modification de tabReel, en temps réel.

➢ *FONCTION avec PARAMETRE en E/S de TYPE POINTEUR :*



DEFINITION :

4. PROTOTYPE fonction (sa DECLARATION), en haut du fichier source :

```
void  NomFonction(TYPE *pt) ;                    // prototype fonction
int main() {    }
```

5. DEFINITION fonction (ses instructions) :

```
void  NomFonction(TYPE *pt) ;                    // prototype fonction
int main() {    }
void  NomFonction(TYPE *pt)                      // définition fonction
{       données et instructions ;
}
```

6. APPEL fonction depuis le main() :

```
void  NomFonction(TYPE *pt) ;                    // prototype fonction
int main()
{    TYPE *pointeur;
     NomFonction(pointeur) ;                     // appel fonction
}
void  NomFonction(TYPE  *pt)                     // définition fonction
{       données et instructions ;                }
```

EXEMPLE :

// Fichier calculMath.cpp

```
void Vabs(int *pta);          // PROTOTYPE - Calcul valeur absolue
_____________________________________________________________________
int main()
{    int  a;

     printf("Donner a : ");          // saisie et réaffichage de l'entier signé
     scanf("%d",&a);
     printf("\na : %d", a);
     Vabs(&a);                       // APPEL - Calcul de sa valeur absolue
     printf("\na : %d", a);           // Affichage de la valeur absolue
}
_____________________________________________________________________
/* DEFINITION - Calcul valeur absolue.      - Paramètre modifié (E/S) : adresse de l'entier
signé (POINTEUR), qui sera remplacé par sa valeur absolue */
void Vabs(int *pta)
{    if  (*pta<0)
     {    *pta= - (*pta);          }
}
```

PRECISION : Un PARAMETRE en ENTREE/SORTIE est **passé par adresse**. La donnée transmise (ex. a) est en lecture/écriture pour la fonction appelée, qui peut la modifier. La fonction a l'accès direct à la zone RAM de a. La modification via pta entraîne la modification de a en temps réel.