Section 2 : Langage C

Partie 2.1 : Introduction au langage C

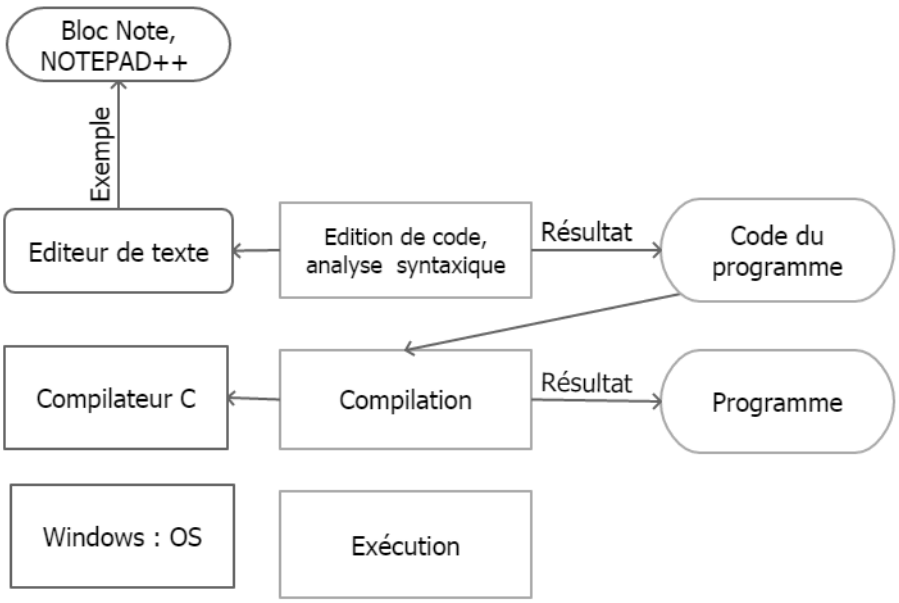
Chapitre 2.1.1 : Bonjour Langage C

Installation du compilateur

- DevC++

https://www.youtube.com/watch?v=Rsuz5XkkoiA

Compilation



TP 1 : Afficher

Ecrire un programme en langage C qui affiche le message « Bonjour langage C »

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

int main(int argc, char *argv[])
{

  printf("Bonjour \n");
  system("PAUSE");
  return 0;
}
```

TP 2 : Lire et écrire un nombre

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

int main(int argc, char *argv[])
{
  int nombre;  // Déclaration d'une variable
  printf("Donnez un nombre : ");
  scanf("%d",&nombre); // Lire (nombre)
  printf("Le nombre que vous avez saisie est : %d",nombre);
  system("PAUSE");
  return 0;
}
```

Chapitre 2.1.2 : Types de base

Notion de type

La **mémoire centrale** est un ensemble de positions binaires nommées **bits**. Les bits sont regroupés en **octets** (8 bits), et chaque octet est repéré par ce qu'on nomme son adresse.

Les types de base du langage C se répartissent en trois grandes catégories en fonction de la nature des informations qu'ils permettent de représenter :

- **nombres entiers** (mot-clé int),
- **nombres flottants** (mot-clé float ou double),
- **caractères** (mot-clé char) ; nous verrons qu'en fait char apparaît (en C) comme un cas particulier de int.
- ~~Chaîne de caractère~~

Types entiers

Le langage C prévoit que, sur une machine donnée, on puisse trouver jusqu'à trois tailles différentes d'entiers, désignées par les mots-clés suivants :

- **short int** (qu'on peut abréger en short),
- **int** (c'est celui que nous avons rencontré dans le chapitre précédent),
- **long int** (qu'on peut abréger en long).

Chaque taille impose naturellement ses limites. Toutefois, ces dernières dépendent, non seulement du mot-clé considéré, mais également de la machine utilisée : tous les `int` n'ont pas la même taille sur toutes les machines ! Fréquemment, deux des trois mots-clés correspondent à une même taille (par exemple, sur PC, `short` et `int` correspondent à 16 bits, tandis que `long` correspond à 32 bits). À titre indicatif, avec 16 bits, on représente des entiers s'étendant de `-32 768` à `32 767` ; avec 32 bits, on peut couvrir les valeurs allant de `-2 147 483 648` à `2 147 483 647`.

TP 2 : Lire et Ecrire un nombre entier

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

int main(int argc, char *argv[])
{
```

```
  int nombre;
  printf("Donnez un nombre : ");
  scanf("%d",&nombre); //Lire(nombre)
  printf("Le nombre que vous avez saisie est : %d",nombre);
  system("PAUSE");
  return 0;
}
```

Types flottants

Les types flottants permettent de représenter, **de manière approchée**, une partie des nombres réels.

```
float nombre ;
scanf("%f",&nombre);
printf("Le nombre que vous avez saisie est : %f",nombre);
```

TP 3 : Lire et écrire un nombre flottant

Ecrire un programme qui lire et écrire un nombre flottants

Types caractères

TP 4 : Lire et écrire un caractère

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

int main(int argc, char *argv[])
{
  char unCaractere ;

  printf("Ecrire un caractère  : ");
  scanf("%c",&unCaractere);
  printf("Le caractère que vous avez saisie est : %c \n",unCaractere);

  system("PAUSE");
  return 0;
}
```

Caractères disposant d'une notation spéciale

NOTATION EN C	CODE ASCII (hexadécimal)	ABRÉVIATION USUELLE	SIGNIFICATION
\a	07	BEL	cloche ou bip (alert ou audible bell)
\b	08	BS	Retour arrière (Backspace)
\f	0C	FF	Saut de page (Form Feed)
\n	0A	LF	Saut de ligne (Line Feed)
\r	0D	CR	Retour chariot (Carriage Return)
\t	09	HT	Tabulation horizontale (Horizontal Tab)
\v	0B	VT	Tabulation verticale (Vertical Tab)
\\	5C	\	
\'	2C	'	
\"	22	"	
\?	3F	?	

Chapitre 2.1.3 : entrées-sorties

Jusqu'ici, nous avons utilisé de façon intuitive les fonctions `printf` et `scanf` pour afficher des informations à l'écran ou pour en lire au clavier. Nous vous proposons maintenant étudier en détail les différentes possibilités de ces fonctions,

Les possibilités de la fonction `printf`

Les principaux codes de conversion

c	`char` : caractère affiché « en clair » (convient aussi à `short` ou à `int` compte tenu des conversions systématiques)
d	`int` (convient aussi à `char` ou à `int`, compte tenu des conversions systématiques)
u	`unsigned int` (convient aussi à `unsigned char` ou à `unsigned short`, compte tenu des conversions systématiques)
ld	`long`
lu	`unsigned long`
f	`double` ou `float` (compte tenu des conversions systématiques `float -> double`) écrit en notation décimale avec six chiffres après le point (par exemple : `1.234500` ou `123.456789`)
e	`double` ou `float` (compte tenu des conversions systématiques `float -> double`) écrit en notation exponentielle (mantisse entre `1` inclus et `10` exclu) avec six chiffres après le point décimal, sous la forme `x.xxxxxxe+yyy` ou `x.xxxxxx-yyy` pour les nombres positifs et `-x.xxxxxxe+yyy` ou `-x.xxxxxxe-yyy` pour les nombres négatifs
s	chaîne de caractères dont on fournit l'adresse (notion qui sera étudiée ultérieurement)

La macro putchar

L'expression :
putchar (a)
joue le même rôle que :
printf ("%c", a)

Les possibilités de la fonction `scanf`

Les principaux codes de conversion de `scanf`

c	char
d	int
u	unsigned int
hd	short int
hu	unsigned short
ld	long int
lu	unsigned long
f ou **e**	loat écrit indifféremment dans l'une des deux notations : décimale (éventuellement sans point, c'est-à-dire comme un entier) ou exponentielle (avec la lettre e ou E)
lf ou **le**	double avec la même présentation que ci-dessus
s	chaîne de caractères dont on fournit l'adresse (notion qui sera étudiée ultérieurement)

Contrairement à ce qui se passait pour printf, il ne peut plus y avoir ici de conversion automatique puisque l'argument transmis à scanf est l'adresse d'un emplacement mémoire. C'est ce qui justifie l'existence d'un code hd par exemple pour le type short ou encore celle des codes lf et le pour le type double.

La macro getchar

L'expression :
a = getchar()
joue le même rôle que :
scanf ("%c", &a)

TP 5 : Possibilités de la fonction « printf »

Quels seront les résultats fournis par ce programme

```
#include <stdio.h>
main ()
{    int n = 543 ;
     int p = 5 ;
     float x = 34.5678;
     printf ("A : %d %f\n", n, x) ;
     printf ("B : %4d %10f\n", n, x) ;
     printf ("C : %2d %3f\n", n, x) ;
     printf ("D : %10.3f %10.3e\n", x, x) ;
     printf ("E : %*d\n", p, n) ;
     printf ("F : %*.*f\n", 12, 5, x) ;
}
```

TP 6 : Possibilités de la fonction « scanf »

Quelles seront les valeurs lues dans les variables *n* et *p* (de type *int*), par l'instruction suivante ?
scanf ("%4d %2d", &n, &p) ;
Lorsqu'on lui fournit les données suivantes (le symbole ^ représente un espace et le symbole @ représente une fin de ligne, c'est-à-dire une validation) ?

a) `12^45@`
b) `123456@`
c) `123456^7@`
d) `1^458@`
e) `^^^4567^^8912@`

Solution des TP

TP1 : Affichage de message bonjour

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

int main(int argc, char *argv[])
{

  printf("Bonjour \n");
  system("PAUSE");
  return 0;
}
```

TP2 : Lire et écrire un nombre entier

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

int main(int argc, char *argv[])
{
  int nombre;
  printf("Donnez un nombre : ");
  scanf("%d",&nombre);

  printf("Le nombre que vous avez saisie est : %d",nombre);

  system("PAUSE");
  return 0;
}
```

TP3 : Lire et écrire un nombre flottante

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

int main(int argc, char *argv[])
{
  float nombre;
  printf("Donnez un nombre : ");
  scanf("%f",&nombre);

  printf("Le nombre que vous avez saisie est : %f",nombre);

  system("PAUSE");
  return 0;
}
```

Partie 2.2 : Instruction de contrôle

Chapitre 2.2.1 : Condition

Rappel :

Valeur booléen
Opérateur booléen

L'instruction `if`

TP 7 : Instruction if

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

int main(int argc, char *argv[])
{
  int unNombre ;

  printf("Donnez un nombre : ");
  scanf("%d",&unNombre);

  if ( unNombre > 0 ){
       printf("Le nombre que vous avez saisie est positive \n");
  } else {
       printf("Le nombre que vous avez saisie est négative \n");
  }

  system("PAUSE");
  return 0;
}
```

TP 8 : étude de cas - Prix TTC

Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement

TP 9 : Afficher la valeur maximale de deux variables

Ecrire un programme qui demande deux valeurs et affiche la valeur maximale de
- Deux nombre
- Deux float
- Deux caractères

TP 10 : étude de cas – Calcule avec If

Le prix de photocopies dans une reprographie varie selon le nombre demandé: 0,5 DH la copie pour un nombre de copies inférieur à 10, 0,4DH pour un nombre compris entre 10 et 20 et 0,3DH au-delà.

TP 11 : équation de deuxième degré

Question 1

```
int main(int argc, char *argv[]) {
   int x =   sqrt(9);
   printf("%d",x);
   return 0;
}
```

Question 2 :

Écrire l'algorithme du traitement qui calcule le discriminant DELTA d'trinôme du second degré AX2 + BX + C et qui, en fonction de son signe, calcule la ou les racines réelles du trinôme ou affiche, si besoin est qu'il n'ya pas de racine réelle.
Les trois coefficients A, B et C seront saisis au clavier avant traitement.

Instruction `switch`

TP 12: Instruction switch

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

int main(){

int n ;
printf ("donnez un entier : ") ;
scanf ("%d", &n) ;
switch (n)
{
      case 0 : printf ("zéro\n") ;
               break ;
      case 1 : printf ("un\n") ;
               break ;
      case 2 : printf ("deux\n") ;
               break ;
}
system("pause");
return 0;
}
```

TP 13 : Instruction switch avec default

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

int main(){

int n ;
printf ("donnez un entier : ") ;
scanf ("%d", &n) ;
switch (n)
{
      case 0 : printf ("zéro\n") ;
               break ;
      case 1 : printf ("un\n") ;
               break ;
      case 2 : printf ("deux\n") ;
               break ;
      default : printf ("grand\n") ;
}
system("pause");
return 0;
}
```

TP 14 : Instruction Switch – rôle de « break »

```c
#include<stdio.h>
#include<stdlib.h>
int main(){

    int n ;
    printf ("donnez un entier : ") ;
    scanf ("%d", &n) ;
    switch (n)
    {
        case 0 : printf ("nul\n") ;
                 break ;
        case 1 :
        case 2 : printf ("petit\n") ;
        case 3 :
        case 4 :
        case 5 : printf ("moyen\n") ;
                 break ;
        default : printf ("grand\n") ;
    }
    system("pause");
    return 0;
}
```

TP 15 : Menu d'application – Calculatrice

Question 1 :

Ecrire un programme qui calcule la somme, la soustraction, la division et le Modulo entre deux variables données par l'utilisateur.

Question 2 :

Ajouter à votre programme un menu d'utilisateur lui permet de choisir l'opération arithmétique à réaliser.
a – en utilisant l'instruction « If »
b – en utilisant l'instruction « Switch »

Exemple d'exécution

```
Les opérations possibles
 + : Addition
 + : Soustraction
 + : Multiplication
 + : Division
 + : Module
Préciser l'opération à réaliser : +
```

```
Donnez la valeur 1 : 4
Donnez la valeur 2 : 2
La somme de 4 et 2 égale : 6
```

Chapitre 2.2.2 : Boucle

L'instruction `for`

Étudions maintenant la dernière instruction permettant de réaliser des boucles, à savoir l'instruction for.

TP 16 : Boucle For

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

int main()
{
  int i ;
  for ( i=1 ; i<=5 ; i++ ) {
      printf ("bonjour ") ;
      printf ("%d fois\n", i) ;
  }
  system("PAUSE");
  return 0;
}
```

L'instruction break

Nous avons déjà vu le rôle de break au sein du bloc régi par une instruction switch. Le langage C autorise également l'emploi de cette instruction dans une boucle. Dans ce cas, elle sert à interrompre le déroulement de la boucle, en passant à l'instruction qui suit cette boucle.

TP 17 : Utilisation de Break dans une boucle

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

int main(int argc, char *argv[])
{

  int i ;
  for ( i=1 ; i<=10 ; i++ ){
      printf ("début tour %d\n", i) ;
      printf ("bonjour\n");
      if ( i==3 ) break ;
         printf ("fin tour %d\n", i) ;
  }

  printf ("après la boucle") ;

  system("PAUSE");
  return 0;
}
```

L'instruction continue

L'instruction continue, quant à elle, permet de passer prématurément au tour de boucle suivant.

TP 18 : Utilisation de continue

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

int main(int argc, char *argv[])
{

  int i ;
  for ( i=1 ; i<=5 ; i++ ){
      printf ("début tour %d\n", i) ;
      if (i<4) continue ;
      printf ("bonjour\n") ;
  }

  system("PAUSE");
  return 0;
}
```

TP 19 : Affichage de la table de multiplication

Question 1 :

Écrire un programme qui affiche la table de multiplication de 9

```
1 * 9 = 1
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
5 * 9 = 45
6 * 9 = 54
7 * 9 = 63
8 * 9 = 72
9 * 9 = 81
```

Question 2 :

Écrire un programme qui affiche la table de multiplication des nombres de 1 à 10, sous la forme suivante :

```
      I   1   2   3   4   5   6   7   8   9  10
-----------------------------------------------
   1  I   1   2   3   4   5   6   7   8   9  10
   2  I   2   4   6   8  10  12  14  16  18  20
   3  I   3   6   9  12  15  18  21  24  27  30
   4  I   4   8  12  16  20  24  28  32  36  40
   5  I   5  10  15  20  25  30  35  40  45  50
   6  I   6  12  18  24  30  36  42  48  54  60
   7  I   7  14  21  28  35  42  49  56  63  70
   8  I   8  16  24  32  40  48  56  64  72  80
   9  I   9  18  27  36  45  54  63  72  81  90
  10  I  10  20  30  40  50  60  70  80  90 100
```

TP 20 : Utilisation des boucles Niveau 2

Afficher un triangle rempli d'étoiles, s'étendant sur un nombre de lignes fourni en donnée et se présentant comme dans cet exemple

```
*
**
***
****
*****
```

TP 21 : Utilisation des boucles Niveau 3

Afficher un triangle rempli d'étoiles, s'étendant sur un nombre de lignes fourni en donnée et se présentant comme dans cet exemple

```
   *
  ***
 *****
*******
```

L'instruction `while`

Voyons maintenant la deuxième façon de réaliser une boucle conditionnelle, à savoir l'instruction « while ».

TP 22 - Cours: Boucle While

Question 1 : que fait l'instruction n+=2

```
int n ;
n = 2 ;
n = n + 2 ;
printf(''%d'',n) ;
n += 2 ;
printf(''%d'',n) ;
```

Question 2 :

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

int main()
{
  int n, somme ;
  somme = 0 ;
  while (somme <100) {
        printf ("donnez un nombre : ") ;
        scanf ("%d", &n) ;
        somme += n ;
  }
  printf ("somme obtenue : %d", somme) ;

  system("PAUSE");
  return 0;
}
```

TP 23 : La somme des nombres

Ecrire un programme qui affiche la somme des nombres saisies par l'utilisateur.
La somme ne doit pas dépasser la valeur 100.

TP 24 : Traduction de for vers while et do..while

Soit le petit programme suivant

```
#include <stdio.h>
main()
{
    int i, n, som ;
    som = 0 ;
    for (i=0 ; i<4 ; i++)
        { printf ("donnez un entier ") ;
          scanf ("%d", &n) ;
          som += n ;
        }
    printf ("Somme : %d\n", som) ;
}
```

Écrire un programme réalisant exactement la même chose, en employant, à la place de l'instruction for :

- une instruction while,
- une instruction do... while.

TP 25: Boucle qui s'arrête à une valeur avec calcule

Calculer la moyenne de notes fournies au clavier avec un dialogue de ce type :
note 1 : 12
note 2 : 15.25
note 3 : 13.5
note 4 : 8.75
note 5 : -1
moyenne de ces 4 notes : 12.37
Le nombre de notes n'est pas connu a priori et l'utilisateur peut en fournir autant qu'il le désire. Pour signaler qu'il a terminé, on convient qu'il fournira une note fictive négative. Celle-ci ne devra naturellement pas être prise en compte dans le calcul de la moyenne.

TP 26 : Déterminer si un nombre est premier

Déterminer si un nombre est un nombre premier ou non.
NB. Un nombre premier est divisible sur 1 et lui même
Question 1
Donnez un nombre : 8
Le nombre 8 n'est pas un nombre premier
Question 2
Donnez un nombre premier : 8
Le nombre 8 n'est pas un nombre premier
Donnez un nombre premier : 7
Oui, 7 est un nombre premier

L'instruction do… while

TP 27 : Boucle - do… while

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

int main()
{
  int n ;
  do {
     printf ("donnez un nb >0 : ") ;
     scanf ("%d", &n) ;
     printf ("vous avez fourni %d\n", n) ;
  } while (n<=0) ;
  printf ("réponse correcte") ;

  system("PAUSE");
  return 0;
}
```

Répète l'instruction qu'elle contient (ici un bloc) tant que la condition mentionnée (`n<=0`) est vraie. On ne sait pas a priori combien de fois une telle boucle sera répétée. Toutefois elle est toujours parcourue au moins une fois. En effet, la condition qui régit cette boucle n'est examinée qu'à la fin de chaque répétition.

TP 28: Menu d'application

Question 1 : Ecrire un programme qui fait l'addition

Donnez X : 10
Donnez Y : 3
Solution de X+Y : 13

Question 2 : Ecrire un programme qui fait la soustraction

Question 3 : Ecrire un programme qui fait les opération arithmétique (+,-,*,/)

Calculatrice
1 – Addition
2 – Soustraction
3 – Division
4 – Multiplication
Donnez votre choix : 2
Donnez X : 10

```
Donnez Y : 3
Solution de X – Y : 7
```

Chapitre : 2.2.3 Opérateurs et expressions

Les conversions implicites pouvant intervenir dans un calcul d'expression

Notion d'expression mixte

TP 29 : Expression Mixte

Question 1 : Exécuter le code suivant

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

int main(int argc, char *argv[])
{
  int nombre_entier;
  float nombre_float ;

  nombre_entier = 1;
  nombre_float = 2.2;

  printf(" Le résultat est  : %f \n", nombre_entier + nombre_float );
  printf(" Le résultat est  : %d \n", nombre_entier + nombre_float );

  system("PAUSE");
  return 0;
}
```

Question 2 : Quel est le type de résultat de l'expression : nombre_entier + nombre_float

Les conversions d'ajustement de type

Une conversion telle que **int -> float** se nomme une « conversion d'ajustement de type ».

Une telle conversion ne peut se faire que suivant une hiérarchie qui permet de ne pas dénaturer la valeur initiale, à savoir :

```
int -> long -> float -> double -> long double
```

On peut bien sûr convertir directement un **int** en **double** ; en revanche, on ne pourra pas convertir un **double** en **float** ou en **int**.
Notez que le choix des conversions à mettre en oeuvre est effectué en considérant un à un les opérandes concernés et non pas l'expression de façon globale. Par exemple, si `n` est de type `int`, `p` de type `long` et `x` de type `float`, l'expression :

```
n * p + x
```

```
   n   *   p   +   x
   |       |       |
 long      |       |     conversion de n en long
   |       |       |
   |__ *  __|       |     multiplication par p
       |           |
     long          |     le résultat de * est de type long
       |           |
     float         |     il est converti en float
       |           |
       |____ +  ___|     pour être additionné à x
            |
          float          ce qui fournit un résultat de type float
```

TP 30 : Conversion implicite

Soit les déclarations suivantes :

```
int n = 10, p = 4 ;
long q = 2 ;
float x = 1.75 ;
```

Question 1 : Donner le type et la valeur de chacune des expressions suivantes :

a) `n + q`
b) `n + x`
c) `n % p +q`
d) `n < p`
e) `n >= p`
f) `n > q`
g) `q + 3 * (n > p)`

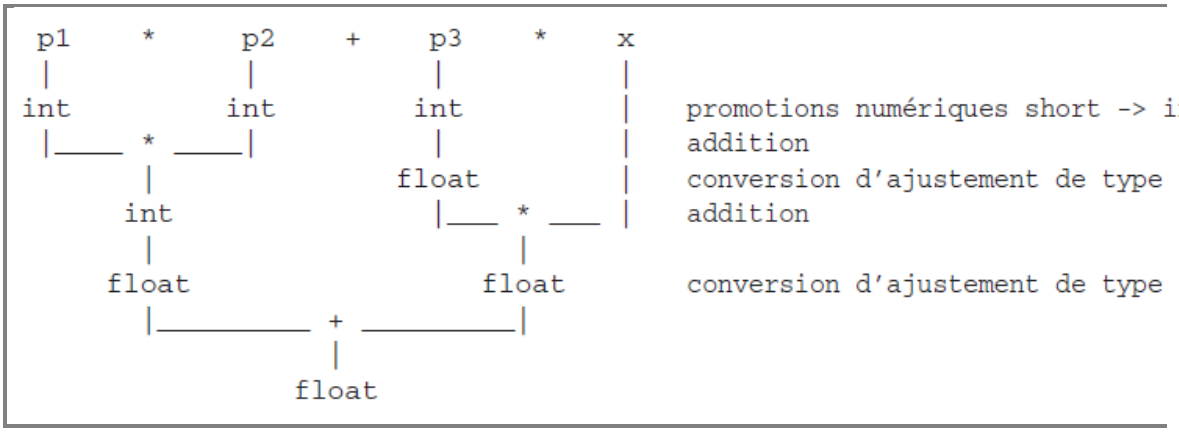
Question 2 : Ecrire un programme qui permet de vérifier les réponse de la question 1

Les promotions numériques

Les conversions d'ajustement de type ne suffisent pas à régler tous les cas. En effet, comme nous l'avons déjà dit, les opérateurs numériques ne sont pas définis pour les types **char** et **short**.
En fait, le langage C prévoit tout simplement que toute valeur de l'un de ces deux types apparaissant dans une expression est d'abord convertie en **int**.
Par exemple, si `p1`, `p2` et `p3` sont de type `short` et `x` de type `float`, l'expression :

```
p1 * p2 + p3 * x
```

est évaluée comme l'indique le schéma ci-après :



Les opérateurs relationnels

TP 31 : Valeur Booléen

Exécuter le code suivant en remarquant la valeur des deux expression suivant :

- a > b
- b > a

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

int main(int argc, char *argv[])
{

  int a,b;
  a = 1;
  b = 2;
  printf("la valeur de a > b = %d \n",a > b );
  printf("la valeur de b > a = %d \n",b > a );
  system("PAUSE");
  return 0;
}
```

Les différents opérateurs relationnels :

OPÉRATEUR	SIGNIFICATION
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
==	égal à
!=	différent de

Le résultat de la comparaison de a > b est, non pas une valeur booléenne (on dit aussi logique) prenant l'une des deux valeurs *vrai* ou *faux*, mais un entier valant :
• 0 si le résultat de la comparaison est faux,
• 1 si le résultat de la comparaison est vrai.

Les opérateurs logiques

C dispose de trois opérateurs logiques classiques : **et** (noté `&&`), **ou** (noté `||`) et **non** (noté `!`).
Par exemple :

`(a<b) && (c<d)` prend la valeur `1` (vrai) si les deux expressions `a<b` et `c<d` sont toutes deux vraies (de valeur `1`), et prend la valeur `0` (faux) dans le cas contraire.

`(a<b) || (c<d)` prend la valeur `1` (vrai) si l'une au moins des deux conditions `a<b` et `c<d` est vraie (de valeur `1`), et prend la valeur `0` (faux) dans le cas contraire

`! (a<b)`	prend la valeur `1` (vrai) si la condition `a<b` est fausse (de valeur `0`) et prend la valeur `0` (faux) dans le cas contraire. Cette expression est équivalente à : `a>=b`.

TP 32 : opérateurs logiques

Exécuter le code suivant en remarquant

- le type de la valeur « a > b && d > c »
- la condition de la structure de contrôle « If »
- Le bloc exécuté de la structure de la structure de contrôle « If » ou « else » et pourquoi ?
- La possibilité d'affectation « resultat_logique = a > b && d > c »

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

int main(int argc, char *argv[])
{
  int a,b,c,d;
  int resultat_logique;
  a = 1; b = 2;c = 3; d = 4;

  printf(" a = %d \n b = %d \n c = %d \n d = %d \n",a,b,c,d);

  printf("la valeur de a > b = %d \n", a > b);
  printf("la valeur de d > c = %d \n", d > c);
  printf("la valeur de a > b && d > c = %d \n", a > b && d > c);

  resultat_logique = a > b && d > c;
  printf("la valeur de a > b && d > c = %d \n", a > b && d > c);

  if (a > b && c > d)
   printf("la valeur de 'a > b && c > d' est différent à 0");
  else
    printf("la valeur de 'a > b && c > d' est égale à 0 \n") ;

  system("PAUSE");
  return 0;
}
```

L'opérateur d'affectation ordinaire
Les opérateurs d'incrémentation et de décrémentation

TP 33 : Pré-Incrémentation et Post-Incrémentation

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int Nombre1 = 0,
      Nombre2=0,
      pre_incrementation = 0,
      post_incrementation = 0;

  printf("Nombre1 = %d \n",Nombre1);
  printf("Nombre2 = %d \n\n",Nombre2);

  pre_incrementation = ++Nombre1;
  post_incrementation = Nombre2++;
  printf("pre_incrementation = %d  \n",pre_incrementation);
  printf("post_incrementation = %d  \n\n",post_incrementation);

  printf("Nombre1 = %d \n",Nombre1);
  printf("Nombre2 = %d \n\n",Nombre2);

  system("PAUSE");
  return 0;
}
```

Les opérateurs d'affectation élargie

+= -= *= /= %=

TP 34 : Opérateur d'affectation élargie

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int Nombre1 = 1,
      Nombre2= 2,
      pre_incrementation = 0,
      post_incrementation = 0;

  printf("Nombre1 = %d \n",Nombre1);
  printf("Nombre2 = %d \n\n",Nombre2);

  Nombre1 += Nombre2;


  printf("Nombre1 = %d \n",Nombre1);
  printf("Nombre2 = %d \n\n",Nombre2);

  system("PAUSE");
  return 0;
}
```

Les conversions forcées par une affectation

TP 35 : Conversion forcée par une affectation

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int NombreInt = 1;
  float   NombreFloat= 2.2;


  printf("NombreInt = %d \n",NombreInt);
  printf("Nombre2 = %f \n\n",NombreFloat);

  NombreInt = NombreFloat + 5;


  printf("Nombre1 = %d \n",NombreInt);
  printf("Nombre2 = %f \n\n",NombreFloat);

  system("PAUSE");
  return 0;
}
```

L'opérateur de cast

S'il le souhaite, le programmeur peut forcer la conversion d'une expression quelconque dans un type de son choix, à l'aide d'un opérateur un peu particulier nommé en anglais « **cast** ».

TP 36 : Utilisation de Cast

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int NombreInt = 1;
  float   NombreFloat= 2.6;


  printf("NombreInt = %d \n",NombreInt);
  printf("Nombre2 = %f \n\n",NombreFloat);

  NombreInt =  NombreFloat + 5.6;
  printf("NombreFloat + 5.6 = %d \n",NombreInt);

  NombreInt = (int) NombreFloat + 5.6;
  printf("Nombre(int) NombreFloatInt = %d \n",NombreInt);

  system("PAUSE");
  return 0;
}
```

L'opérateur conditionnel

Considérons l'instruction suivante :

```
if ( a>b )
        max = a ;
   else
        max = b ;
```

Elle attribue à la variable `max` la plus grande des deux valeurs de `a` et de `b`. La valeur de `max`
Pourrait être définie par cette phrase :

```
Si a>b alors a sinon b
```

En langage C, il est possible, grâce à l'aide de **l'opérateur conditionnel**, de traduire presque littéralement la phrase ci-dessus de la manière suivante :

```
max = a>b ? a : b
```

TP 37 : Opérateur conditionnel

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  float NombreInt = 2;
  float   NombreFloat = 2.6;
  float max = 0;

  max = NombreFloat > NombreInt ? NombreFloat:NombreInt;

  printf("max = %f \n",max);

  system("PAUSE");
  return 0;
}
```

L'opérateur `sizeof`

L'opérateur `sizeof`, dont l'emploi ressemble à celui d'une fonction, fournit la taille en octets
(n'oubliez pas que l'octet est, en fait, la plus petite partie adressable de la mémoire).

TP 38 : Opérateur - sizeof

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

int main(int argc, char *argv[])
{
  int nombreInt = 2;
  long nombreLong = 200000000;
  float nombreFloat = 2.2;
  double nombreDouble = 2.00000002;
  char caractere = 'A' ;

  printf("sizeof (nombreInt) = %d \n", sizeof(nombreInt));
  printf("sizeof (nombreLong) = %d \n", sizeof(nombreLong));
  printf("sizeof (nombreFloat) = %d \n", sizeof(nombreFloat));
  printf("sizeof (nombreDouble) = %d \n", sizeof(nombreDouble));
  printf("sizeof (caractere) = %d \n", sizeof(caractere));

  system("PAUSE");
  return 0;
}
```

TP 39 : sur l'opérateur conditionnel

Écrire plus simplement l'instruction suivante

```c
z = (a>b ? a : b) + (a <= b ? a : b) ;
```

TP 40 : sur l'opérateur conditionnel

n étant de type **int**, écrire une expression qui prend la valeur :

- -1 si n est négatif,
- 0 si n est 0,
- 1 si n est positif.

TP 41 : Exercice sur l'opérateur d'incrémentation

Quels résultats fournit le programme suivant ?

```c
#include <stdio.h>
main()
{
   int n=10, p=5, q=10, r ;
   r = n == (p = q) ;
   printf ("A : n = %d  p = %d  q = %d  r = %d\n", n, p, q, r) ;
   n = p = q = 5 ;
   n += p += q ;
   printf ("B : n = %d  p = %d  q = %d\n", n, p, q) ;
   q = n < p ? n++ : p++ ;
   printf ("C : n = %d  p = %d  q = %d\n", n, p, q) ;
   q = n > p ? n++ : p++ ;
   printf ("D : n = %d  p = %d  q = %d\n", n, p, q) ;
}
```