Travaux Pratiques N° 3
Entrées/sorties et préprocesseur

But du TP :

- Introduire les instructions d'entrée/sorties.
- Se familiariser avec quelques directives de base du Préprocesseur ;
- Comprendre suffisamment `scanf()` pour ne pas être surpris par son comportement surtout en lisant des données de type différents.
- Ce TP suppose des connaissances en représentation des nombres et des caractères (voir cours de Stucture Machine 1).

Énoncé :

A. Entrées/sorties de base et quelques spécificateurs de formats

La fonction printf est une fonction d'écriture formatée vers la sortie standard (l'écran). Le but de cette fonction est d'afficher une chaîne de caractères ou la valeur d'une variable selon un format à spécifier. Les indicateurs de format couramment utilisés sont :

%d	entier relatif
%f	réel
%u	entier naturel (unsigned i.e. non signé)
%c	caractère
%s	chaîne de caractères (nous l'utiliserons dans un chapitre à venir)

```c
int a=8;
float b1=2.990812,b3=2.990813;
double b2=2.990812;

char c='A';

int main(){
   printf("1. Les entiers:");
   printf("\na=%d a=%+d -a=%+d  donc + pour afficher le signe",a,a,-a);
   printf("\na=%5d a=%+5d -a=%+5d pour afficher sur un nombre de positions ici 5",a,a,-a);
   printf("\na=%05d a=%+05d -a=%+05d pour completer les positions vides par des zeros",a,a,-a);
   printf("\na=%5d a=%-5d -a=%-5d donc - pour justifier a gauche",a,a,-a);
   printf("\na=%i une autre possibilité pour les entiers",a);
   printf("\na=%o pour afficher %d en octal",a,a);
   printf("\n3*a=%x pour afficher %d en hexadecimal",3*a,3*a);
   printf("\n\n2. Les reels:");
   printf("\nb1=%f donc le nombre affiché est tronqué",b1);
   printf("\nb1=%e b1/100=%e la notation scientifique",b1,b1/100);
   printf("\nb1=%.4f pour imposer une précision ici 4 chiffres apres la virgule",b1);
   printf("\nb1=%.1f avec un chiffre décimal, le nombre a changé",b1);
   printf("\nb1=%.4f la valeur n'a pas ete modifiee en memoire",b1);
   printf("\nb1=%6.3f pour afficher sur un nombre de positions avec une précision",b1);
   printf("\nb1=%6.2f pour afficher sur un nombre de positions (ici 6) le point décimal est inclus",b1);
   printf("\nb2=%lf pour afficher un réel double précision",b2);
   b1=0.2;b3=0.200001;b2=0.2;
   if(b1!=b3){
      printf("\nMalgre les apparences, b1=%.4f et b3=%.4f sont differents",b1,b3);
   }
   //attendre le ch representation des réels (SM) pour comprendre
   if(b1!=b2){
      printf("\nMalgre les apparences, b1=%f et b2=%lf sont differents",b1,b2);
   }
   printf("\n\n3. Les caracteres:");
   printf("\nc=%c affiche le caractère",c);
   printf("\nc=%d affiche son code ASCII",c);
   return 0;
}
```

- Quels sont les types et les valeurs d'initialisation des variables `b1, b2` et `b3` ?
- Quels sont les différents spécificateurs de format que ce code contient ?
- Exécuter le programme. Essayer de comprendre son comportement en vous aidant des messages affichés.

```
1. Les entiers:
a=8 a=+8 -a=-8  donc + pour afficher le signe
a=    8 a=   +8 -a=   -8 pour afficher sur un nombre de positions ici 5
a=00008 a=+0008 -a=-0008 pour completer les positions vides par des zeros
a=    8 a=8     -a=-8    donc - pour justifier a gauche
a=8 une autre possibilité pour les entiers
a=10 pour afficher 8 en octal
3*a=18 pour afficher 24 en hexadecimal

2. Les reels:
b1=2.990812 donc le nombre affiché est tronqué
b1=2.990812e+00 b1/100=2.990812e-02 la notation scientifique
b1=2.9908 pour imposer une précision ici 4 chiffres apres la virgule
b1=3.0 avec un chiffre décimal, le nombre a changé
b1=2.9908 la valeur n'a pas ete modifiee en memoire
b1= 2.991 pour afficher sur un nombre de positions avec une précision
b1=  2.99 pour afficher sur un nombre de positions (ici 6) le point décimal est
inclus
b2=2.990812 pour afficher un réel double précision
Malgre les apparences, b1=0.2000 et b3=0.2000 sont differents
Malgre les apparences, b1=0.200000 et b2=0.200000 sont differents

3. Les caracteres:
c=A affiche le caractère
c=65 affiche son code ASCII
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

- Quels sont les cas où la valeur affichée est différente du contenu initial de la variable ? Pourquoi il y a différence ?
- Pourquoi accède-t-on aux instructions conditionnelles affichant « `Malgre les apparences ...` » ?
- Quel est le rôle de \n ? Essayer avec \t et déduire son rôle.

La fonction `scanf` est utilisée pour lire des données depuis le clavier. Comme `printf`, ou presque, `scanf` nécessite un indicateur de format, suivi d'une liste d'arguments à lire. Le flux de caractères entré au clavier est mis dans un buffer (il y a un seul buffer commun à tous vos `scanf`). `scanf` retire ces caractères un par un tant qu'elle est en mesure de réaliser une correspondance avec le format spécifié sinon elle les laisse dans le buffer (les blancs en début de lecture de numériques sont retirés et ignorés). On peut utiliser la valeur retournée par `scanf` pour savoir si la lecture a été effectuée avec succès.

Voici un exemple instructif utilisant une boucle `do while()` qui sert à répéter l'exécution des instructions entre `do` et `while()`, (on y reviendra dans un prochain TP). Prendre son temps à l'exécuter avec différents inputs (voir la capture d'exécution ci-après). L'exécution est réalisée sous ubuntu. Avec d'autres systèmes d'exploitation, elle peut être différente.

```c
int a,b,nbrLect;
char myPurgeCar;
int main(){
    printf("entrer deux valeurs entieres: ");
    nbrLect=scanf("%d %d",&a,&b);
    printf("J'ai pu lire les valeurs %d et %d i.e. %d en tout",a,b,nbrLect);
    if(nbrLect!=2){
       printf("\nDonc il y eu un problème. Essayons de voir ce qu'il y'a"
              " dans le buffer");
    }
    do{
        scanf("%c",&myPurgeCar);
        printf("\nLe buffer contient le caractere de code "
               "ASCII %d qui est -->%c<--",myPurgeCar,myPurgeCar);
    }while(myPurgeCar!='\n');
    return 0;
}
```

1. Si on entre 10 puis espace puis 30 :

```
entrer deux valeurs entieres: 10 30
J'ai pu lire les valeurs 10 et 30 i.e. 2 en tout
Le buffer contient le caractere de code ASCII 10 qui est -->
<--
Process returned 0 (0x0)   execution time : 3.544 s
Press ENTER to continue.
```

- Pourquoi les 2 dernières flèches ne sont pas sur la même ligne ?

2. Si on entre 10 puis espace,'e','t',espace puis 30 :

```
entrer deux valeurs entieres: 10 et 30
J'ai pu lire les valeurs 10 et 0 i.e. 1 en tout
Donc il y eu un problème. Essayons de voir ce qu'il y'a dans le buffer
Le buffer contient le caractere de code ASCII 101 qui est -->e<--
Le buffer contient le caractere de code ASCII 116 qui est -->t<--
Le buffer contient le caractere de code ASCII 32 qui est --> <--
Le buffer contient le caractere de code ASCII 51 qui est -->3<--
Le buffer contient le caractere de code ASCII 48 qui est -->0<--
Le buffer contient le caractere de code ASCII 10 qui est -->
<--
Process returned 0 (0x0)   execution time : 5.491 s
```

- Nous avons entré 2 espaces. Le premier avant 'et' mais l'autre après. Pourquoi on ne trouve pas le premier dans le buffer à la différence du deuxième (Le buffer contient … ASCII 32 ...) ?

3. Si on entre 1,0,espace,5,0,espace,6,0,merci puis on tape entrée :

```
entrer deux valeurs entieres: 10 50 60 merci
J'ai pu lire les valeurs 10 et 50 i.e. 2 en tout
Le buffer contient le caractere de code ASCII 32 qui est --> <--
Le buffer contient le caractere de code ASCII 54 qui est -->6<--
Le buffer contient le caractere de code ASCII 48 qui est -->0<--
Le buffer contient le caractere de code ASCII 32 qui est --> <--
Le buffer contient le caractere de code ASCII 109 qui est -->m<--
Le buffer contient le caractere de code ASCII 101 qui est -->e<--
Le buffer contient le caractere de code ASCII 114 qui est -->r<--
Le buffer contient le caractere de code ASCII 99 qui est -->c<--
Le buffer contient le caractere de code ASCII 105 qui est -->i<--
Le buffer contient le caractere de code ASCII 10 qui est -->
<--
```

- Pourquoi ici l'espace avant le caractère 6 est laissé dans le buffer et non éliminé.

Rem : Ces exécutions ont été réalisées sous Linux. La table ASCII est sur la dernière page.
Un autre point à retenir : pour lire une variable scalaire avec scanf, il faut la préfixer par l'opérateur unaire & (i.e. adresse de) car scanf a besoin de l'adresse mémoire des données à lire.
Voici un autre exemple :

- Exécuter le programme suivant avec une seule valeur, deux valeurs puis trois avec et sans /.

```c
#include <stdio.h>
#include <stdlib.h>
int j,m,a;
int main(){
    printf("entrer une date avec le format jj/mm/aaaa \n");
    scanf("%d/%d/%d",&j,&m,&a);
    printf("J'ai pu lire la date  %02d/%02d/%04d \n",j,m,a);
    return 0;
}
```

En voici quelques exécutions :

```
entrer une date avec le format jj/mm/aaaa
12
J'ai pu lire la date  12/00/0000
```

```
entrer une date avec le format jj/mm/aaaa
12/10
J'ai pu lire la date  12/10/0000
```

```
entrer une date avec le format jj/mm/aaaa
05/11/2017
J'ai pu lire la date  05/11/2017
```

```
entrer une date avec le format jj/mm/aaaa
12 4 2013
J'ai pu lire la date  12/00/0000
```

- Qu'est ce que vous remarquez ? Expliquer à la lumière de ce que vous avez appris sur `scanf`.

B. Préprocesseur (include et define)
Le préprocesseur est un programme exécuté avant la compilation. Il modifie le fichier source à partir de directives introduites par le caractère #. Dans ce TP, on s'intéresse à quelques fonctionnalités pour effectuer les opérations suivantes :

- l'ajout de fichiers source avec #include,
- la définition de constantes symboliques et de macros avec #define,

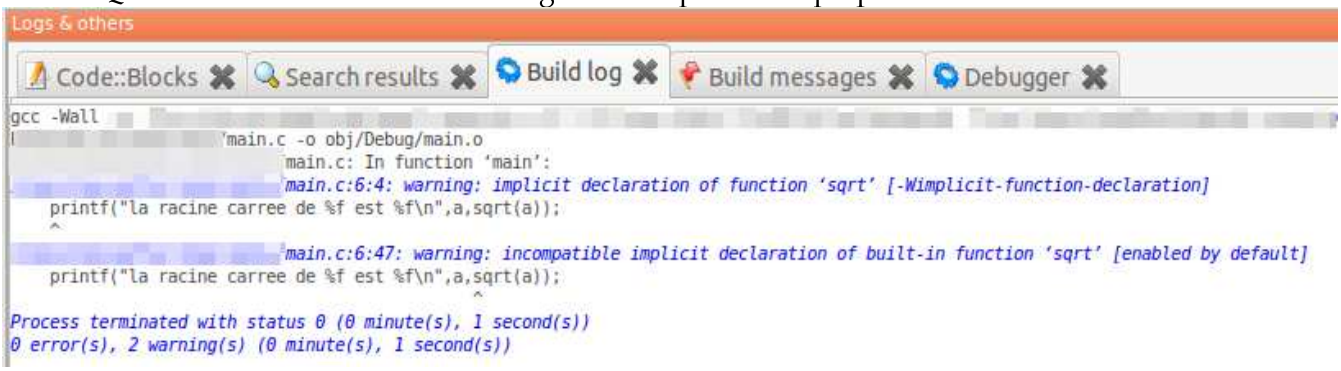
2.1 La directive #include : permet d'ajouter dans le fichier source le texte figurant dans un autre fichier. On l'utilisera pour ajouter un fichier en-tête tel que stdio.h, math.h, etc. pour pouvoir utiliser des fonctions prédéfinies telles que printf(), scanf(), sqrt(), etc.

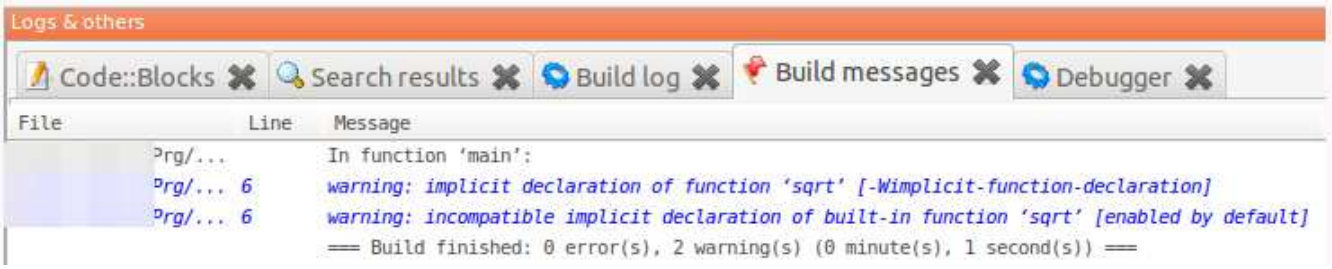
Saisir le programme 3 suivant et l'exécuter avec et sans la ligne 2.

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

int main(){
    float a=64;
    printf("la racine carree de %f est %f\n",a,sqrt(a));
    return 0;
}
```

- Que notez-vous ? Visualiser les logs suivant puis les expliquer.





2.2 La directive #define : On l'utilisera pour définir des constantes symboliques. Sa syntaxe est :

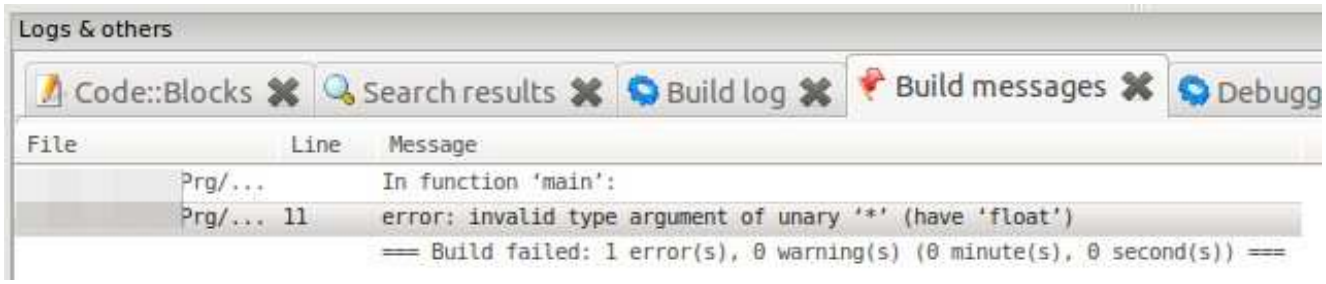
#define <identifiant de la constante> <valeur>

Elle demande au préprocesseur de remplacer toute apparition de <identifiant de la constante> par la valeur <valeur>. Son utilité est de pouvoir facilement modifier la valeur de la constante sans avoir à parcourir tout le fichier (On l'utilisera aussi pour indiquer la taille de tableau mais il faut attendre le chapitre 3 pour comprendre cette notion de tableau).

Le programme 4 essaye de calculer la circonférence d'un cercle étant donné son rayon.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define PI 3.14;
5   float surf,circonf,r;
6
7   int main(){
8       printf("Donner le rayon!\n");
9       scanf("%f",&r);
10      circonf=2*r*PI;
11      surf=PI*r*r;
12      return 0;
13  }
```

- Le reprendre tel qu'il est, puis le compiler.
- Pourquoi il y a erreur dans la ligne 11 ?
- Pourquoi la ligne 10 ne génère pas d'erreur ?
- Pourquoi le compilateur pense que '*' est ici un opérateur unaire alors que le programmeur veut utiliser un opérateur de multiplication binaire ?



- Corrigez-le en tenant compte du message du compilateur.
- Ajouter les instructions pour afficher la surface et la circonférence.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]