Introduction à la Programmation C

Licence L1 – MASS

Informatique S1

Prof. Responsable :

Manuele Kirsch Pinheiro

Manuele.Kirsch-Pinheiro@univ-paris1.fr

http://crinfo.univ-paris1.fr/users/manuele/manuele.htm

Université Paris 1 – Panthéon Sorbonne
L1 – MASS
Informatique S1

Informatique S1
Programmation C

- Objectifs
  - introduction à l'algorithmique
  - Introduction au langage C
- Organisation
  - CM : 12 séances de 1h
  - TD : 12 séances de 2h
    - **Présence obligatoire sur les TDs**
- Évaluation
  - 50% examen, 50% contrôle continu
  - Contrôle continu (TD)
    - Interrogations surprises (QCM)
    - Devoir long
    - Participation

http://epi.univ-paris1.fr/masslic1c

Algorithmique

- Algorithme
  - Suite finie des pas à effectuer, dans un ordre donnée, afin de parvenir à un résultat
  - Actions pour résoudre un problème
  - Exemple : Changer une ampoule
    - Quels sont les pas à effectuer pour changer une ampoule ?

Algorithme

- Programme
  - Implémentation d'un algorithme à l'aide d'un langage de programmation
    - L'algorithme est la « recette » du programme

Langage C

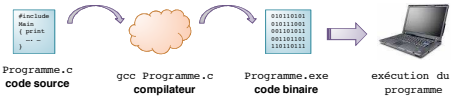
- Historique
  - Création en 1972 par Denis Ritchie (entre autres) avec l'objectif d'écrire un système d'exploitation (Unix, le « grand-père » de Linux)
  - Résultat : un langage performant et versatile

Langage C

- Évolution
  - Le langage C a bien évolué dans le temps
  - 78 : Kernighan & Ritchie
  - Années 80 : standardisation ANSI C

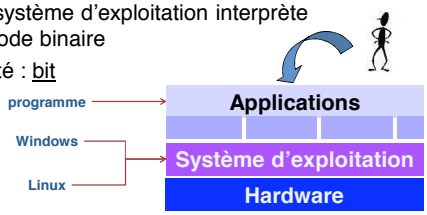
Langage C

- Comment ça marche ?!
  - Le C est un langage compilé
    1) On écrit le programme (code source) avec un éditer de texte non-formaté
    2) On passe le code source au compilateur
    3) On exécute le code binaire (le programme)



Exécution d'un programme

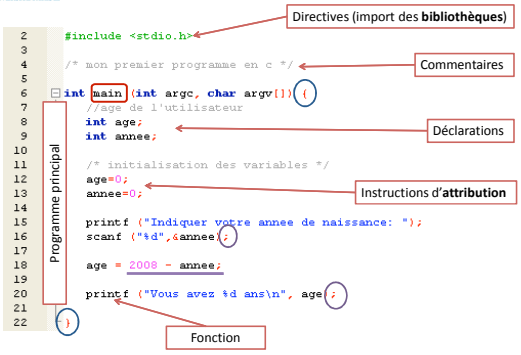
- Code binaire
  - Le « langage » compris par les machines
  - Le système d'exploitation interprète le code binaire
  - Unité : bit



Informatique S1
Programmation C

- *Objectifs de la séance*
  - structure générale d'un programme en C
- *Concepts*
  - Structure générale
  - Mots-clés
  - Variables
  - Opérateurs arithmétiques
  - Commentaires
- Exemples avec *printf* et *scanf*

Structure général



Mots-clés

- Mots réservées
- Mots reconnues par le compilateur
- Chacune a une signification particulière

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Variables

- Définition :
  - Une entité qui contient une information
  - Les variables sont stockées dans la mémoire
- Caractéristiques :
  - nom → identifiant
  - valeur
  - type
    - Le type indique l'ensemble des valeurs que peut prendre la variable

Les types des données

- Les principaux types acceptés en langage C :
  - Entiers
    - `int` (± 16 bits)
    - `short` (± 8 bits)
    - `long` (± 32bits)
  - Réels
    - `float`
    - `double`
  - Caractères
    - `char`
  - `Signed / unsigned`
  - `void`

char	float	int
long	short	signed
unsigned	void	

Actions sur les variables

- Que peut-on faire avec une variable ?
  - Déclaration → définir la variable
  - Lire (obtenir) sa valeur → regarder son contenu
  - Affecter une (nouvelle) valeur → mettre une (nouvelle) information
    - Opérateur d'attribution : « = »

Déclaration d'une variable

- Format :
  - type identifiant *[, identifiant…] [= valeur initial]* ;
- Exemples :

```
int   i, j, k;
char  lettre;
float  reel01 = 1.25;
```

optionnel

On sépare les identifiants par les « , »

type

identifiant

Identifiant
lettre [lettres, chiffres ou _]
a10 ~~10a~~
var_int ~~var!~~

Attribution

- Opérateur d'attribution : « **=** »
- Exemple :

```
c = a + b;
```

- Évaluation
  - On prend la valeur contenue dans la variable **a**
  - On prend la valeur contenue dans la variable **b**
  - On additionne (opérateur « + ») ces deux valeurs
  - On met ce résultat dans la variable **c**
- Si c avait auparavant une valeur, cette dernière est **perdue** !

Opérateurs

- Opérateurs arithmétiques :

```
*     /
%
+     -
```

Ordre de priorité

Le résultat d'un opérateur dépend des types des variables !!

- Exemples :

```
int a, b;
a = 2 + 3 * 5;
b = 2 * 5 + 5 % 2;
b = 5 / 2;
```

25 ou 17 ?

Exemple

```
2   #include <stdio.h>
3
4   /* mon premier programme en c */
5
6   int main (int argc, char argv[]) {
7       //age de l'utilisateur
8       int age;
9       int annee;
10
11      /* initialisation des variables */
12      age=0;
13      annee=0;
14
15      printf ("Indiquer votre annee de naissance: ");
16      scanf ("%d",&annee);
17
18      age = 2008 - annee;
19
20      printf ("Vous avez %d ans\n", age);
21
22  }
```

scanf ("%d", &annee);

int → %d

printf ("Votre age %d \n", age);

Entrée & Sortie

- Entrée formatée : `scanf`
- Sortie formatée : `printf`
- Formats
  - `%d` `int`
  - `%i`
  - `%f` `float`
  - `%e`
  - `%c` `char`

printf ("Texte - format ", variables);

scanf ("format", &variables);

printf ("%d %f", var_int, var_float);

- Caractères spéciaux
  - `\n` nouvelle ligne
  - `\t` tab
  - `\\` la « \ »
  - `\"` le « " »
  - `%%` le « % »

Informatique S1
Programmation C

- *Objectifs de la séance*
  - Entrées et sorties
- *Concepts*
  - Sortie formatée avec *printf*
  - Entrée formatée avec *scanf*

Entrée & Sortie en C

- E/S garantit la communication avec l'utilisateur
  - Entrée : demander à l'utilisateur de lui fournir une information
  - Sortie : présenter (afficher) une information à l'utilisateur
- Plusieurs fonctions disponibles
  - printf
  - scanf

Sortie : printf

```
#include <stdio.h>

/* mon premier programme en c */

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

   printf ("Hello World!\n");

}
```

```
kirsch@vbxlinux: ~/code
Fichier  Édition  Affichage  Terminal  Onglets  Aide
kirsch@vbxlinux:~/code$ gcc -o td1 td1.c
kirsch@vbxlinux:~/code$ ./td1
Hello World!
kirsch@vbxlinux:~/code$
```

Sortie : printf

- Sortie formatée
- Formats
  - `%d` `int`
  - `%i`
  - `%f` `float`
  - `%e`
  - `%c` `char`

printf ("Texte - format ", variables);

printf ("%d %f", var_int, var_float);

- Caractères spéciaux
  - `\n` nouvelle ligne
  - `\t` tab
  - `\\` la « \ »
  - `\"` le « " »
  - `%%` le « % »

Exemple

```
#include <stdio.h>

/* TD format sortie
 * Objectif : presenter differents format de sortie avec printf
 */

int main (int argc, char argv[]) {
    int entier = 1;
    float reel = 1.2345;  /* attention : c'est "." et non "," */
    char lettre = 'a';

    printf ("Une bonne presentation :\n");
    printf ("  entier=%d    reel=%f  char=%c\n"  entier, reel, lettre);
    printf ("  entier=%i    reel=%e  char=%c\n"  entier, reel, lettre);

}
```

```
kirsch@vbxlinux: ~/code
Fichier  Edition  Affichage  Terminal  Onglets  Aide
kirsch@vbxlinux:~/code$ gcc -o sortie sortie.c
kirsch@vbxlinux:~/code$ ./sortie
Une bonne presentation :
  entier=1    reel=1.234500   char=a
  entier=1    reel=1.234500e+00  char=a
kirsch@vbxlinux:~/code$
```

Entrée : scanf

```
2   #include <stdio.h>
3
4   /* mon premier programme en c */
5
6   int main (int argc, char argv[]) {
7       //age de l'utilisateur
8       int age;
9       int annee;
10
11      /* initialisation des variables */
12      age=0;
13      annee=0;
14
15      printf ("Indiquer votre annee de naissance: ");
16      scanf ("%d",&annee);
17
18      age = 2008 - annee;
19
20      printf ("Vous avez %d ans\n", age);
21
22  }
```

Lire une valeur au clavier

`scanf ("%d", &annee);`

int → %d

`printf ("Votre age %d \n", age);`

Entrée : scanf

- Entrée formatée : `scanf`
- Formats
  - `%d` `int`
  - `%ld` `long`
  - `%f` `float`
  - `%e`
  - `%lf` `double`
  - `%le`
  - `%c` `char`

`scanf ("format", &variable);`

`scanf ("%d", &var_int);`

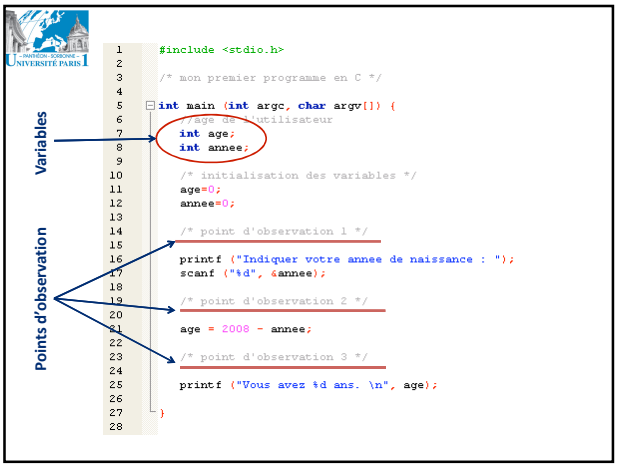
&
Le **&** garantit que la valeur obtenue sera bien enregistrée dans la variable

```
#include <stdio.h>

int main () {
    int entier;   /* un nombre entier */
    float reel;   /* un nombre reel */
    char lettre;  /* une lettre */

    printf ("Entrer une lettre : ");
    scanf ("%c",&lettre);
    printf ("Vous avez entre : \t %c \n", lettre);

    printf ("Entrer un nombre entier : ");
    scanf ("%d",&entier);
    printf ("Vous avez entre : \t %d \n",entier);

    printf ("Entrer un nombre reel : ");
    scanf ("%f",&reel);
    printf ("Vous avez entre : \t %f \n",reel);

}
```

```
kirsch@vbxlinux: ~/code
Fichier  Edition  Affichage  Terminal  Onglets  Aide
kirsch@vbxlinux:~/code$ gcc -o sortie sortie.c
kirsch@vbxlinux:~/code$ ./sortie
Entrer une lettre : A
Vous avez entre :        A
Entrer un nombre entier : 2
Vous avez entre :        2
Entrer un nombre reel : 3.45
Vous avez entre :        3.450000
kirsch@vbxlinux:~/code$
```

Tracer l'exécution d'un programme

- Tracer :
  - Simuler sur le papier l'exécution d'un programme
  - Observer le comportement des variables
- Intérêt :
  - Débugger le code
- Méthode :
  - Définition des points d'observation
  - Table variables X point d'observation



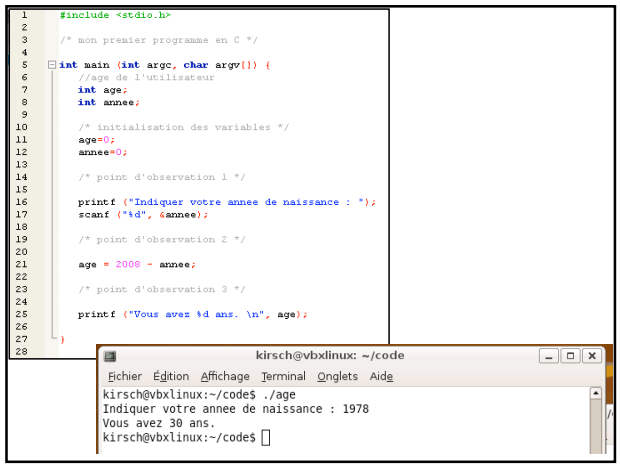
Tracer un programme

Variables

Point d'obs. \ variables	age	annee
Point d'observation 1	0	0
Point d'observation 2	0	1978
Point d'observation 3	30	1978

Points d'observation

On annote dans le tableau la valeur de chaque variable à chaque point d'observation



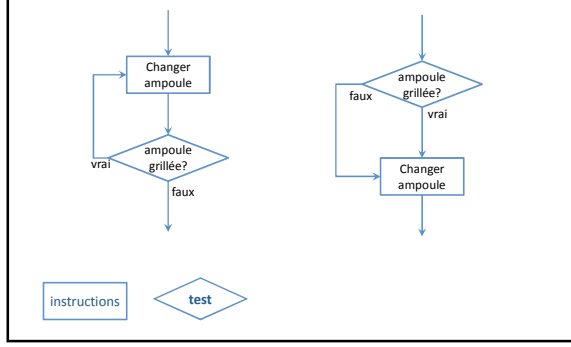
Informatique S1
Programmation C

- *Objectifs de la séance*
  - **Présentation des boucles (Partie I)**
- *Concepts*
  - Opérateurs relationnels >,<,>=,<=, !=, ==
  - Boucle while
  - Boucle do-while
- *Concepts complémentaires*
  - Notion de typecast
  - Indentation

Boucles

- Boucles
  - Répéter un bloc d'instructions tant qu'une condition logique soit vrai
  - On ne sait pas à priori combien de fois le bloc doit être exécuté
- Exemples d'usage :
  - Calculer x^Y avec X et Y fournis par l'utilisateur
  - Calculer le factoriel de n (1 x 2 x … x n)
  - Calculer la moyenne d'un ensemble
  - …

Boucles



Opérateurs relationnels

- Pour comparer deux valeurs

Vrai ou faux
Pas de type booléen en C
Faux → 0 (zero)
Vrai → ≠ 0

==	égal à	`a == b`
!=	différent de	`a != b`
<	inférieur à	`a < b`
<=	inférieur ou égal à	`a <= b`
>	supérieur à	`a > b`
>=	supérieur ou égal à	`a >= b`

Boucle while

- Format :

  while (*test*)

  {

  bloc d'instructions ;

  }

Boucle infinie

Le test doit être vrai à un moment donné ou on restera bloqué !!

```
while (i < y) {
      expo = expo * x;
      i = i + 1;
}
```

Boucle while

```
1  #include <stdio.h>
2
3  int main (int argc, char argv[]) {
4      int x, y;
5      float expo;
6      int i;
7
8      /* point d'observation 1 */
9      printf ("Entrez x : ");
10     scanf ("%d", &x);
11     printf ("Entrez y : ");
12     scanf ("%d", &y);
13
14     expo = 1;
15     i = 0;
16
17     /* point d'observation 2 */
18
19     while (i < y) {
20         expo = expo * x;
21         i = i + 1;
22     }
23
24     /* point d'observation 4 */
25
26     printf ("x ^ y = %f \n", expo);
27 }
```

Test : i < y

instructions

Trace d'exécution				
	expo	i	y	x
PO 2	1	0	2	2
PO 3	2	1	2	2
PO 3	4	2	2	2
PO 4	4	2	2	2

Boucle do-while

- Format :

  do

  {

  bloc d'instructions ;

  } while (*test*) ;

Boucle infinie

Le test doit être vrai à un moment donné ou on restera bloqué !!

```
do {
      expo = expo * x;
      i = i + 1;
} while (i < y);
```

Boucle do-while

```
1  #include <stdio.h>
2
3  int main (int argc, char argv[]) {
4      int x, y;
5      float expo;
6      int i;
7
8      /* point d'observation 1 */
9      printf ("Entrez x : ");
10     scanf ("%d", &x);
11     printf ("Entrez y : ");
12     scanf ("%d", &y);
13
14     expo = 1;
15     i = 0;
16
17     /* point d'observation 2 */
18
19     do {
20         expo = expo * x;
21         i = i + 1;
22     } while (i < y);
23
24     /* point d'observation 4 */
25
26     printf ("x ^ y = %f \n", expo);
27 }
```

instructions

Test : i < y

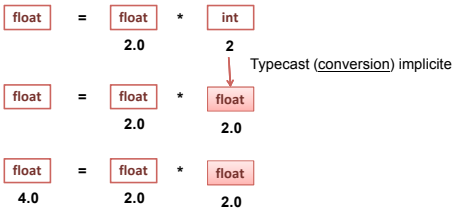
Trace d'exécution				
	expo	i	y	x
PO 2	1	0	2	2
PO 3	2	1	2	2
PO 3	4	2	2	2
PO 4	4	2	2	2

Typecast

- Conversion de type
  - int → float 3 → 3.0
  - float → int 3.5 → 3
- Implicite
  - Automatique
  - Pas assez fiable
- **Explicite**
  - On indique explicitement le nouveau type
  - **(newtype)**

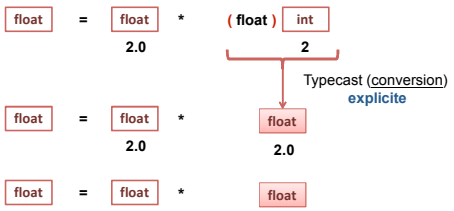
Typecast implicite

```
int x, y;
float expo;
...
    expo = expo * x ;
...
```



Typecast explicite

```
int x, y;
float expo;
...
    expo = expo * (float) x ;
...
```



Typecast explicite

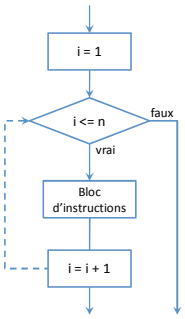
- Avantage :
  - Visibilité

```
int  i = 4;
float x = 3.45;


i = x;
```

```
int  i = 4;
float x = 3.45;


i = (float) x;
```

Boucle *for*

- **Instruction de contrôle** à l'instar de *while* et *do…while*
- Dans une seule instruction :
  - Affectation : i=1
  - Test : i <= n
  - Incrément : i = i + 1



Boucle *for*

```
1  #include <stdio.h>
2
3  /*  Factoriel avec for  */
4
5  int main () {
6
7     int i, n;
8     int factoriel;
9
10     /* point d'observation 1 */
11     printf ("Entrez un nb entier : ");
12     scanf ("%d", &n);
13
14     factoriel = 1;
15
16     /* point d'observation 2 */
17     for (i=1; i<= n; i=i+1) {
18        factoriel = factoriel * i;
19        /* point d'observation 3 */
20     }
21
22     /* point d'observation 4 */
23     printf ("factoriel de %d est %d \n", n, factoriel);
24  }
25
```

Trace d'exécution

	i	n	factoriel
PO 1	?	?	?
PO 2	?	3	1
PO 3	1	3	1
PO 3	2	3	2
PO 3	3	3	6
PO 4	4	3	6

for (i=1 ; i <= n ; i = i+1)

Boucle *for*

- Format :

```
for ( affectation ; test ; incrément )
{
    bloc d'instructions ;
}
```

Affectation test Incrément

```
for (i=1; i<= n; i=i+1) {
   factoriel = factoriel * i;
}
```

Boucles imbriquées

- Une boucle peut contenir une autre…
- Exemples :

```
while (test 1) {
   for (affectation;  test 2; incr)
        {  instructions for;  }
   autres instruction while;
}
```

```
for (affectation;  test 2; incr) {
   while (test 1)
      {  instructions while;  }
   autres instruction for;
}
```

```
while (test 1) {
   while (test 2)
        {  instructions while;  }
   autres instruction while;
}
```

```c
#include <stdio.h>

/*  CM 5: afficher le tableau de multiplication */

int main () {
   int i,j;

   for (i=1; i <= 10; i++) {
      printf ("%d : ", i);
      for (j=1; j <= 10 ; j++) {
         printf ("\t %d", (i*j));
      }
      putchar ('\n');
   }
}
```

à chaque interaction du premier *for*, on exécute tout le second *for*.

```
kirsch@vbxlinux:~/code$ ./cm5-loopimbr
1 :     1       2       3       4       5       6       7       8       9       10
2 :     2       4       6       8       10      12      14      16      18      20
3 :     3       6       9       12      15      18      21      24      27      30
4 :     4       8       12      16      20      24      28      32      36      40
5 :     5       10      15      20      25      30      35      40      45      50
6 :     6       12      18      24      30      36      42      48      54      60
7 :     7       14      21      28      35      42      49      56      63      70
8 :     8       16      24      32      40      48      56      64      72      80
9 :     9       18      27      36      45      54      63      72      81      90
10 :    10      20      30      40      50      60      70      80      90      100
```

E / S non-formaté

```c
#include <stdio.h>

/*  CM 5: afficher le tableau de multiplication */

int main () {
   int i,j;

   for (i=1; i <= 10; i++) {
      printf ("%d : ", i);
      for (j=1; j <= 10 ; j
         printf ("\t %d",
      }
      putchar ('\n');
   }
}
```

Fonctions de la bibliothèque **stdio.h**

- E/S pour 1 caractère
- Sortie :
  `putchar (char)`
- Entrée :
  `char = getchar()`

Attention à l'indentation

```c
#include <stdio.h>

/*  CM 5: afficher le tableau de multiplication */

int main () {
int i,j;

   for (i=1; i <= 10; i++) {
printf ("%d : ", i);
      for (j=1; j <= 10 ; j++) {
printf ("\t %d", (i*j));
      }
putchar ('\n');
}
   }
```

Indentation !!
But : Rendre le code compréhensible

Opérateurs unaires ++ et --

- Instructions très souvent utilisée : i = i + 1;
- Opérateur d'incrémentation **++** :

i ++ ; équivaut i = i + 1;

- Opérateur de décrémentation **--** :

i -- ; équivaut i = i - 1;

Opérateurs ++ et --

```
...
 int i, k;
 i = 0;
 k = 0

 i++;
 printf ("i=%d",i);
...
```

i ++ ; ⇒ i = i + 1;

i = 1

i++ ou ++i ?!

```
...
 i = 3;

 k = i++ - 5;

 printf ("k=%d",k);
...
```

k = i ++ - 5; ⇒
k = i - 5;
i = i + 1;

k = -2
i = 4

i ++
++ après, incrément réalisé après !!!

Opérateurs ++ et --

```
...
 i = 1;

 k = i++;

 printf ("k=%d",k);
...
```

k = i ++ ; ⇒
k = i;
i = i + 1;

k = 1
i = 2

```
...
 i = 2;
 k = ++i;
...
```

k = 2

i++ ou ++i ?!

```
...
 i = 4;

 k = ++i + 5;

 printf ("k=%d",k);
...
```

k = ++i + 5; ⇒
i = i + 1;
k = i + 5;

k = 10
i = 5

++i
++ avant, incrément réalisé avant !!!

Opérateurs d'affectation élargie

- Principaux opérateurs d'affectation :

  += -= *= /= %=

```
k += i;   ⇒  k = k + i;
k -= i;   ⇒  k = k - i;
k *= i;   ⇒  k = k * i;
k /= i;   ⇒  k = k / i;
k %= i;   ⇒  k = k % i;
```

Priorité entre les opérateurs

Opérateurs unaires :	**++**	**--**	**(cast)**	
Opérateurs arithmétiques	*	/	%	
	+	-		
Opérateurs relationnels :	<	<=	>	>=
	==	!=		
Opérateurs logiques :	&&			
	||			
Opérateurs d'affectation :	=	*=	/=	%=
	+=	-=		

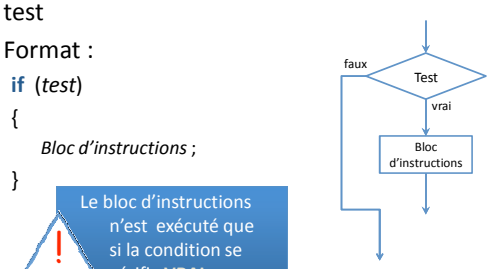
Informatique S1
Programmation C

- *Objectifs* : réaliser des tests simples et emboités
- Présentation de l'instruction if :
  - *if* et *if … else*
- Usage de tests emboités
- Opérateurs logiques
  - && , ||

Instruction *if*

- Instruction de contrôle permettant de faire un test
- Format :

if (*test*)
{
Bloc d'instructions ;
}



Le bloc d'instructions n'est exécuté que si la condition se vérifie **VRAI**

Exemple

```
1   #include <stdio.h>
2
3   /*  CM 6 : instructions de controle if */
4
5   int main () {
6       int k = 0;
7
8       printf ("Entrez k : ");
9       scanf ("%d", &k);
10
11      if (k < 0) {
12         printf ("k est negatif (%d)\n", k);
13      }
14  }
```

Le *printf* ne sera exécuté que si **k < 0**

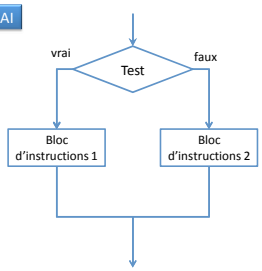
Instruction *if … else*

- Format :

if (*test*)
{
Bloc d'instructions 1;
}
else
{
Bloc d'instructions 2;
}

Exécuté si le test est VRAI

Exécuté si le test est FAUX



Exemple

```
1   #include <stdio.h>
2
3   /*  CM 6 : instructions de controle if */
4
5   int main () {
6       int k = 0;
7
8       printf ("Entrez k : ");
9       scanf ("%d", &k);
10
11      if (k < 0) {
12          printf ("k est negatif (%d)\n", k);
13      }
14      else {
15          printf ("k est positif (%d)\n", k);
16      }
17
18
19  }
```

Si **k < 0**, le *printf* à la ligne **12** s'affiche (" k est négatif ")

Sinon , le *printf* à la ligne **15** s'affiche (" k est positif ")

Instructions *if* emboitées

- Une instruction if peut contenir d'autres instructions de contrôle
  - *if, while, do…while, for…*

```
if (test 1) {
   if (test 2)
        {  instructions if 2;  }
   else
        { instructions else if 2; }
   …
}
else {
   instructions else if 1;
   …
}
```

```
if (test 1) {
   instructions if 1;
}
else {
   …
   if (test 2)
        {  instructions if 2;  }
   else
        { instructions else if 2; }
   …
}
```

Exemple

```
1   #include <stdio.h>
2
3   /*  CM 6 : instructions de controle if */
4
5   int main () {
6       int k = 0;
7
8       printf ("Entrez k : ");
9       scanf ("%d", &k);
10
11      if (k < 0) {
12          printf ("k est negatif (%d)\n", k);
13      }
14      else {
15          printf ("k est positif (%d)\n", k);
16
17          if (k < 5)
18              printf ("\t k est inferieur a 5 \n");
19          else
20              printf ("\t k est superieur ou egal a 5 \n");
21      }
22  }
```

On n'atteint le 2° *if* que si **k < 0** est faux

Opérateur logiques

- Opérateurs de la logique booléenne :
  - Operateur **&&** ⇒ ET (AND)

    `(a => 0)` **&&** `(a < 5)`
  - Opérateur **||** ⇒ OU (OR)

    `(c == 'o')` **||** `(c == 'n')`
  - Opérateur **!** ⇒ NON (NOT)

    ! `(c == 'o')`

Opérateurs logiques

&&	Vrai	Faux
Vrai	V	F
Faux	F	F

a = 0 , b = 1, c = -1

(a < b) && (b>c) ?

(a < b) || (b < c) ?

||	Vrai	Faux
Vrai	V	V
Faux	V	F

!	Vrai	Faux
	F	V

! (a < b) ?

Exemple

```
1  #include <stdio.h>
2
3  /*  CM 6 : instructions de controle if */
4
5  int main () {
6      int k = 0;
7
8      printf ("Entrez k : ");
9      scanf ("%d", &k);
10
11     if ( (k >= 0) && (k <= 5) ) {
12        printf ("k est positif et inferieur (ou egal) a 5 (%d)\n", k);
13     }
14     else if ( (k > 5) ) {
15        printf ("k est positif et superieur a 5 (%d)\n", k);
16     }
17     else {
18        printf ("k est negatif (%d)\n", k);
19     }
20 }
```

(k >= 0) && (k<=5)
Valeur de k entre 0 et 5

Informatique S1
Programmation C

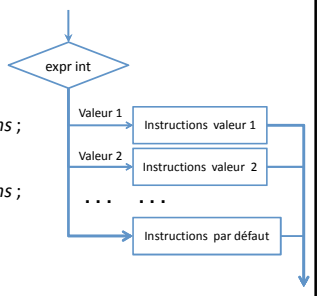
- *Objectifs* : instruction de contrôle *switch*

- Présentation de l'instruction *switch*
- Usage de l'instruction *break*
- Opérateur *? :*

Instruction de contrôle switch

- Test multiples valeurs
- Format :

```
switch (expression_int)
{
  case valeur1 : instructions ;
                 break ;
  case valeur2 : instructions ;
                 break ;
  ...
  default : instructions ;
}
```




```
switch (expr_int)
{
  case valeur1 : instructions ;
                 break ;
  case valeur2 : instructions ;
                 break ;
  ...
  default : instructions ;
}
```

expression résultant un int
- variable de type
  - int
  - char (typecast)
  - long, short, unsigned
- expression

si expr_int == valeur1

break ;
termine l'exécution d'un bloc d'instructions

default (optionnel) :
aucun des cas précédents n'est vrai

Exemple

```
#include <stdio.h>

int main () {
    char c;

    printf ("Entrez une lettre : ");
    scanf ("%c", &c);

    switch (c) {
        case 'a':
                printf ("voyelle A \n");
                break;
        case 'e':
                printf ("voyelle E \n");
                break;
        case 'i':
                printf ("voyelle I \n");
                break;
        case 'o':
                printf ("voyelle O \n");
                break;
        case 'u':
                printf ("voyelle U \n");
                break;
        default:
                printf ("%c n'est pas une voyelle\n",c );
    }
}
```

char c;
. . .
switch (c)

Typecast char ↔ int

break ;
S'il n'y a pas un break ?!

default ;
S'il n'y a pas un default?!

Exemple

```c
#include <stdio.h>
#include <stdlib.h>     //fonction rand
#include <time.h>       //fonction time

/* cm7 : swith (devinette) */
int main () {
   int i,x;
   srand (time(NULL)); //initialise le generateur
   x = rand()%11;

   printf ("Entrez i [0-10] : ");
   scanf ("%d", &i);

   switch (x-i) {
      case 0:   printf ("bingo\n");
                break;
      case 1:
      case -1:
                printf ("presque\n");
                break;
      case 2:
      case -2:
      case 3:
      case -3:  printf ("proche\n");
                break;
      default:  printf ("loin\n");
   }
   printf ("Reponse : %d\n",x);
}
```

Instruction **printf** réalisée dans les deux cas :
(x-i) == 1
(x-i) == -1

Opérateur « ? : »

- Opérateur de test « ? : »

  (*test*) ? *si_vrai* : *si_faux* ;

- À la place de :

```
if (a < b)
      petit = a;
else
      petit = b;
```

- On peut écrire :

```
petit = (a < b) ? a : b ;
```

Exemple

```c
#include <stdio.h>
/*  cm 7 : operateur ?:   */
int main () {
   int a, b, petit, grand;

   printf ("Entrez a : ");
   scanf ("%d", &a);
   printf ("Entrez b : ");
   scanf ("%d", &b);

   petit = (a<b) ? a : b;
   grand = (a>b) ? a : b;

   printf ("%d < %d\n", petit, grand);
}
```

```
if (a < b)
    petit = a;
else
    petit = b;
```

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

Instructions de contrôle

Tests

- if … else
- switch … case … default

Boucles

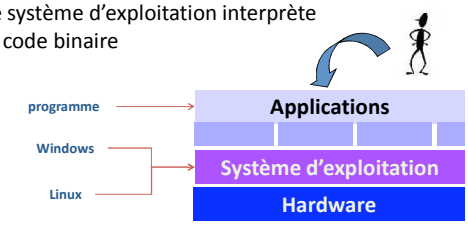
- while
- do … while
- for

Informatique S1
Programmation C

- *Objectifs* :
  - Introduction système binaire
  - Représentation des caractères

Exécution d'un programme

- Code binaire
  - Le « langage » compris par les machines
  - Le système d'exploitation interprète le code binaire



Codage binaire

- Toutes communications à l'intérieur de l'ordinateur sont faites avec des signaux électriques
- Pour simplicité et fiabilité, ces signaux ont deux états seulement :
  - 0 – éteint (absence de signal électrique)
  - 1 – allumé (présence de signal électrique)
- Une **unité d'information** (**0** ou **1**) est appelée **bit** (de l'anglais *binary digit*)

Décimal X Binaire

- Le système décimal
  - Représentation : 10 symboles différents
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Représentation d'un numéro (580) :
    - **5** centaines, **8** dizaines, **0** unités
  - Équivalent mathématique :
    - $\mathbf{5}x10^2 + \mathbf{8}x10^1 + \mathbf{0}x10^0$

Décimal X Binaire

- Système binaire :
  - Représentation : 2 symboles différents
    - **0 (faux)** et **1 (vrai)**
  - Représentation d'un numéro (6) :
    - **110** → **1**x4 + **1**x2 + **0**x1
  - Équivalent mathématique :
    - **1**x2^2 + **1**x2^1 + **0**x2^0

Bits, Bytes, octets, etc...

- ***bit*** *– une unité binaire (0 ou 1)*
- ***octet*** *(ou* ***Byte****) – groupe de 8 bits*
- ***Kilo-octets*** *(Ko) – 1024 octets*
- ***Méga-octets*** *(Mo) – 1024Ko - 1048576 octets*
- ***Giga-octets*** *(Go) – 1024 Mo - 1073741824 octets*

- *Pourquoi 1Ko ≠ 1000 octets?*
  - *Encore, à cause de la base binaire*
  - *2^{10} = 1024*

Taille des types des données

- **int**
  - 16bits → 2^{16} possibilités
  - de **-32768** à **+32767**
- **short int**
  - 8bits → 2^8 possibilités
  - de **-128** à **+127**
- **long int**
  - 32bits → 2^{32} possibilités
  - de **-2147483648** à **+2147483647**

unsigned
(sans signal)
- unsigned short
  - de 0 à 255

Taille des types des données

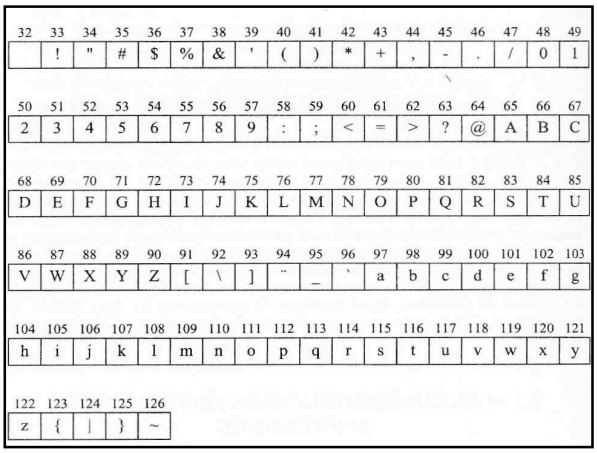
- **float**
  - 32 bits
  - Jusqu'à 3.4e+38
- **double**
  - 64 bits
- **char**
  - 8 bits
  - **256 caractères**

Représentation du type char

- La représentation des caractères par une séquence de bits comme pour les numéros entiers
  - C'est l'indication du type des données qui permet de faire la différence entre eux
- Il faut **coder** les caractères (chiffres, lettres et autres symboles) sous un **format** qui peut être reconnu par tous les ordinateurs
  - **ASCII**
  - UNICODE

ASCII

- Norme internationale pour la représentation des caractères
- 256 caractères y sont représentés
  - Chiffres 0 à 9
  - Lettres de l'alphabet en majuscule et minuscule
  - Caractères spéciaux
    - Space, *, /, \, <, >, !, ?, etc.
  - Caractères de contrôle
    - Nouvelle ligne, bip, tabulation, etc.



32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1

50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
2	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C

68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U

86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103
V	W	X	Y	Z	[	\	]	¨	_	`	a	b	c	d	e	f	g

104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121
h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

122	123	124	125	126
z	{	|	}	~

Type char en langage C

- Représentation selon le code **ASCII**
- char **c = 'A'**;
  correspond à (int) c == 65
- char **c = '0'**;
  correspond à (int) c == 48

Rappel !
Pour mentionner un char, on utilise ' ' :
'a', 'b', '0', '1', '\n', '\t'…

Informatique S1
Programmation C

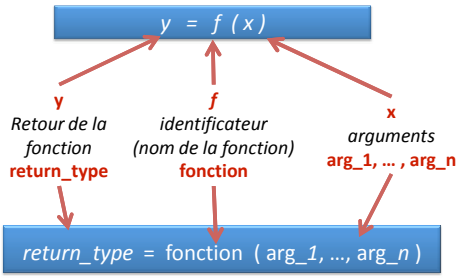
- Objectifs :
  - Usage des fonctions
  - Définition des constantes

Fonctions

- Définition
  - Une fonction est un module constitué d'une suite d'instructions
  - Une fonction est un morceau de code réutilisable
- Exemples

```
main ()
printf ("Hello world !")
c = getchar()
```

Fonctions



Usage des fonctions

- On *appelle* une fonction par son nom, en indiquant les *arguments* appropriés
- Exemples :

```
racine = sqrt ( x );
```

Fonction : « **sqrt** »
Argument : « **x** »
Retour sur la variable « **racine** »

```
printf ("x = %f ", x);
```

Fonction : « **printf** »
Arguments : « **x = %f** » et « **x** »
Retour ignoré

```
c = getchar ();
```

Fonction : « **getchar** »
Argument : pas d'argument
Retour sur la variable « **c** »

Exemple

```
5   /* cm7 : swith (devinette) */
6   int main () {
7       int i,x;
8       srand (time(NULL)); //initialise le generateur
9       x = rand()%11;
10
11      printf ("Entrez [0-10] : ");
12      scanf ("%d", &i);
13
```

time (NULL)

x = rand () % 11

printf ("... ")
scanf ("%d", &i)

srand (...)

Fonction main

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

Retour
int

Arguments
int argc
char argv[]

Bibliothèques

- Les fonctions peuvent être organisées dans les bibliothèques
- Bibliothèques → ensemble de fonctions
- Bibliothèques standard

stdio.h	*stdlib.h*	*math.h*
time.h	*ctype.h*	*limits.h*
string.h	*errno.h*	*float.h*
locale.h	*stdarg.h*	*stddef.h* . . .

Bibliothèques

```
1   #include <stdio.h>
2   #include <stdlib.h>     //fonction rand
3   #include <time.h>       //fonction time
4
5   /* cm7 : swith (devinette) */
6   int main () {
7       int i,x;
8       srand (time(NULL)); //initialise le generateur
9       x = rand()%11;
10
11      printf ("Entrez [
12      scanf ("%d", &i);
13
```

#include <*stdlib.h*>
↳ import de la bibliothèque *stdlib.h*

Les fonctions de *stdlib.h* sont disponibles :
- *srand* (...)
- *rand* ()
- *abs* ()
- ...

Pour compiler
gcc **-lm** -o prog prog.c

Quelques fonctions utiles…

- stdio.h
  - int printf (char* format, ...)
  - int scanf (char* format, ...)
  - int getchar (void)
  - int putchar (int c)
- stdlib.h
  - int abs (int n)
  - long labs (long n)
  - int rand (**void**)
  - void srand (unsigned int *seed*)
- ctype.h
  - int isalpha (int *c*)
  - int isupper(int *c*)
  - int islower(int *c*)
  - int iscntrl (int *c*
  - int isdigit (int *c*)
  - int isalnum (int *c*)
  - int tolower(int *c*)
  - int toupper(int *c*)
- math.h
  - double pow(double *x*, double *y*)
  - double sqrt(double *x*)
  - double log10(double *x*)
  - double fabs(double *x*)
  - double sin(double *x*)

void
Type de donnés
indique pas d'argument ou pas de retour

Définition d'une constante

- Les bibliothèques peuvent contenir la définition des constantes
- constante → valeur ne change pas

M_PI
NULL
FLT_MAX
DBL_MAX
SHRT_MAX
INT_MIN
…

x = pow (M_PI, 2);
srand (time(NULL));

Définition d'une constante

- Définition

`#define CTE valeur`

Directive pré-compilation

gcc remplace touts les « CTE » par « valeur »

- Exemples

```
#define   MAX   10
#define   FAUX  0
#define   VRAI  1
```

Habituellement, on utilise des MAJUSCULES pour les noms des constantes

Exemple

```
1   #include <stdio.h>
2   #define  MIN  -15
3   #define  MAX  45
4
5
6   /* Conversion C-F */
7   main () {
8     float fahr;
9     int celc;
10
11
12    printf ("Celsius\tFahrenheit \n");
13
14    for (celc=MIN; celc <= MAX; celc+=5) {
15        fahr = ( (double)celc * 9)/5 + 32;
16        printf (" %d \t %.2f\n", celc, fahr);
17    }
18
19  }
```

Définition
#define MIN -15
#define MAX 45

Usage
celc = MIN;
celc <= MAX;

Informatique S1
Programmation C

- Objectifs :
  - Introduction à la notion d'array unidimensionnel

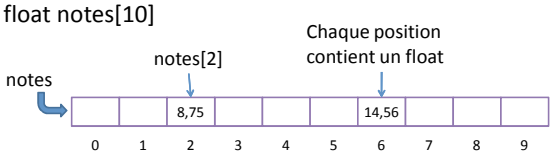
Array / Vecteur

- Problème :
  - On veut calculer la moyenne d'un groupe et afficher les notes inférieures à la moyenne
  - Il faut garder les notes afin de les comparer à la moyenne
- Solutions :
  - Solution 1 :
    - Avoir autant de variables que d'étudiants dans le groupe
    - Pas satisfaisant !!
      - Si le groupe contient 150 étudiants ? 350 ? 500 ??
  - Solution 2 : un **vecteur** !!

Array / Vecteur

- Un **array** (ou un vecteur) est une structure de données contentant un **ensemble de données** d'un **même type**.
- Un vecteur est une **variable** qui contient plusieurs espaces pour garder les **valeurs**

float notes[10]



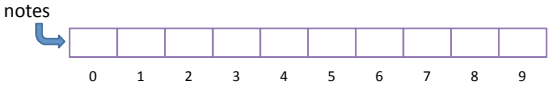
Déclaration d'un array

float notes [10] ;

Type des donnés

Identificateur (nom de la variable)

N° de positions (taille)



Déclaration avec initialisation

float notes [10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} ;

- Type des donnés
- Identificateur (nom de la variable)
- N° de positions (taille)
- Valeurs

Attention à ne pas fournir plus des valeurs que de positions

notes

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Usage d'un vecteur

float notes [**10**] ; — taille

notes [2] = 10.25;

- Indice N° entier → Indice de **0** à **9** (taille – 1)

notes[2]

notes

0	1	2	3	4	5	6	7	8	9
		10.25							

Usage d'un vecteur

- L'indice est un n° entier (**int**)
  - note [2] note [1 + 1]
  - note [i] note [i – 1]
- Exemple

```
11  for (i=0; i<MAX; i++) {
12      printf ("Entrez note %d : ",i);
13      scanf ("%f", &notes[i]);
14
15      somme += notes[i];
16  }
```

scanf ("%f", **&notes[i]**);

somme += **notes[i]**;

Attention à ne pas dépasser la limite (la taille du vecteur) !!!

Exemple

```
1   #include <stdio.h>
2   #define MAX  10
3
4   /* cm 10 : array */
5
6   int main () {
7       float somme, moy;
8       float notes[MAX];
9       int i;
10
11      for (i=0; i<MAX; i++) {
12          printf ("Entrez note %d : ",i);
13          scanf ("%f", &notes[i]);
14
15          somme += notes[i];
16      }
17
18      moy = somme / MAX;
19
20      printf ("Moyenne du groupe : %.2f \n", moy);
21      printf ("Notes inferieures a la moyenne : \n");
22
23      for (i=0; i<MAX; i++) {
24          if (notes[i]<moy) {
25              printf ("Note %d: %.2f \n", i, notes[i]);
26          }
27      }
28
29  }
```

#define MAX 10

float notes[MAX];

scanf ("%f", &notes[i]);
somme += notes[i];

if (notes[i] < moy)

Exercices

Informatique S1 – Programmation C
Exercices

TD 1 : Prise en main de l'environnement

Dans ce TD, nous allons faire des exercices pour la prise en main de l'environnement sous Linux utilisé dans les salles machines. Pour y accéder, nous avons besoin d'un identifiant et d'un mot de passe. Dans notre cas, il s'agit de l'identifiant « user », sans aucun mot de passe.

Exercice 1

a) Trouver l'éditeur de texte (*gedit*) dans le menu d'applications. Où l'avez-vous trouvé ?

b) Taper le texte suivant (attention à bien respecter les espaces et les signes de ponctuation)

```
#include <stdio.h>

/* mon premier programme en c */

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

   printf ("Hello World!\n");

   return (0);
}
```

c) Enregistrer le texte dans un fichier nommé « helloword.c ». Dans quel répertoire l'avez-vous enregistré ?

d) Toujours dans l'éditeur de texte, aller sur le menu[1] « affichage → Modes de coloration → Sources » et choisir l'option « C ». Que se passe-t-il ?

e) Trouver le navigateur Web. Entrer dans son courrier électronique de l'Université Paris 1, et envoyer à vous-même le fichier helloword.c (en tant que pièce jointe).

Exercice 2

Les exercices suivants doivent être réalisés à l'aide des informations contenues dans le guide de référence rapide qui vous a été fourni.

a) Trouver dans le menu d'applications le terminal. Lancer-le.

b) Indiquer dans quel répertoire êtes-vous. Donner le chemin complet.

c) Afficher le contenu du répertoire courant. Quelle commande avez-vous utilisée pour le faire ?

d) Créer un répertoire nommé « code » dans le répertoire courant.

e) Entrer dans le nouveau répertoire « code », exécuter la commande « ls -la ». Que représente-t-elle ?

f) Quel est le chemin complet du répertoire « code ».

g) Exécuter la commande « cd .. » suivie de la commande « pwd ». Que se passe-t-il ? Qu'avait fait la première commande ?

[1] On suppose ici un éditeur en Français. Il se peut qu'il soit en Anglais. Dans ce cas, chercher l'équivalent dans le menu « view ».

h) Aller dans le répertoire où vous avez enregistré le ficher « helloworld.c ». Taper la commande « cat HelloWorld.c ». Que se passe-t-il ? Et la commande « cat helloworld.c » ?

i) Copier le fichier « helloword.c » vers un fichier « HelloWorld.c ». Quelle commande avez-vous utilisée ?

j) Refaire la question 2-h). Que se contient le fichier « HelloWorld.c » ?

k) Déplacer le fichier « HelloWorld.c » vers le répertoire « code » (utiliser pour cela le chemin complet de ce répertoire).

l) À l'aide de la commande « rmdir », essayer d'effacer le répertoire « code ». Que se passe-t-il ? Comment devons-nous procéder afin d'effacer ce répertoire ?

Exercice 3

a) Brancher une clé USB sur l'ordinateur. Que se passe-t-il ? Essayer de localiser, à l'aide de la commande « ls », le répertoire représentant la clé USB (essayer sous les répertoires « /media » ou « /mnt »).

b) Copier le répertoire « code » et son contenu (astuce : utiliser la commande « man » pour comprendre comment faire ça avec la commande « cp »).

c) Déconnecter la clé USB.

Exercice 4

a) Comparer la sortie des commandes « which gcc » et « whereis gcc ». Quelle est la principale différence entre ces deux commandes ?

b) Exécuter les commandes suivantes « cat helloworld.c > test_redir.txt », suivie de « cat helloworld.c >> test_redir.txt ». Que contiendra le document test_redir.txt une fois ces commandes exécutées ?

c) Comment pouvons-nous écrire la commande cat helloworld.c >> test_redir.txt sur deux lignes (par exemple, le mot « cat » dans une ligne et le reste « helloworld.c >> test_redir.txt » dans la ligne suivante ? Expliquer.

d) Comparer les commandes « cat », « more » et « less », en exécutant « cat test_redir.txt », « more test_redir.txt » et « less test_redir.txt ».

e) Que fait la commande « cat test_redir.txt | more » ? Pouvons-nous faire la même chose en utilisant la commande « less » à la place de la commande « more » ?

f) Que fait la commande « clear » ? Et la commande « echo » ? Quelles sont leurs options ? (Astuce : utiliser la commande « man »)

g) Que fait la commande « ls *.c » ? Que présente-t-elle ?

h) Archiver les fichiers créés au cours de la séance à l'aide de la commande « zip ».

i) À l'aide de l'éditeur de texte *gedit* (vu au début de la séance), modifier le fichier « test_redir.txt » en incluant la ligne « /* CECI EST UN TEST */ » à la ligne 11.

j) Lister les fichiers inclus dans le nouveau fichier « zip » à l'aide de la commande « unzip ». Quelle option avez-vous utilisé ? Si vous exécuter la commande « unzip » sans cette option, qu'arrivera-t-il à vos fichiers ?

Exercice 5

a) Aller dans le répertoire où se trouve le fichier « helloworld.c », et taper la commande « gcc helloworld.c ». Que se passe-t-il ? Maintenant, taper la commande « gcc –o helloworld helloworld.c ». Que se passe-t-il ? (Astuce : utiliser la commande « ls »).

Informatique S1 – Programmation C
Exercices

TD 2 : Structure générale d'un programme C

Dans ce TD, nous allons réaliser des exercices couvrant la structure générale d'un programme en C et l'usage des variables. Nous allons faire des programmes simples en utilisant les concepts vus en cours.

Compilation

Avant de commencer les exercices, il faut savoir comment compiler ses programmes. Pour compiler un programme en C, on utilisera le compilateur « gcc » (Gnu C). Il faut donc ouvrir un terminal, aller sur le répertoire où se trouvent le (ou les) source(s) qu'on veut compiler, et taper la commande « gcc –o source source.c » (en remplaçant le « source.c » par le nom du fichier qu'on veut compiler). Par exemple :

```
gcc –o helloworld helloworld.c
```

Puis, pour exécuter le programme, il suffit de taper :

```
./helloworld
```

(remplacer le « helloworld » par le nom indiquer avec l'option « -o » de la commande gcc)

Exercice 1

a) Le programme ci-dessous présente plusieurs erreurs. Trouvez-les.

```c
#include <stdio.h>

main ()
  int myInt1, myInt2;

  printf ("Entrer le nombre d'heures : ")
  scanf ("%d", myInt1);

  myint2 = myint1 * 3600;

  printf ("Il y a %d s en %d h " myInt1, myInt2);
}
```

b) Analyser le programme ci-dessus. Décrire que devrait faire chaque ligne.

c) Corriger le programme ci-dessus et l'exécuter.

Exercice 2

Écrire un programme en langage C qui affiche le message suivant :

```
Bonjour, entrer un nombre entier s'il vous plait :
```

L'utilisateur doit alors entrer un nombre entier (par exemple 72), et le programme doit répondre avec le même numéro que l'utilisateur a fourni :

```
Le nombre que vous avez entre est 72
```

Exercice 3

Écrire un programme en langage C qui lit un nombre entier fourni par l'utilisateur, calcule et affiche à l'utilisateur le *double* de ce nombre. Comment faire pour que le programme affiche également le triple ? (Astuce : pour lire le nombre, utiliser « scanf » vu en cours).

Exercice 4

a) Écrire un programme qui demande à l'utilisateur deux nombres entiers et qui affiche la somme entre les deux nombres.

b) Étendre le programme précédent pour qu'il affiche également la soustraction entre les deux nombres lus (premier – second).

c) Étendre encore le programme précédent pour qu'il affiche la division (premier / second) et le reste de la division (premier % second).

d) À partir de la dernière version du programme précédent, que se passe-t-il si l'utilisateur introduit les nombres « 9 » et « 2 » respectivement ? Quelles réponses aurons-nous pour la division et le reste ? Comment faire pour afficher le résultat de la division lorsque ce résultat n'est pas un nombre entier ?

e) Que se passe-t-il pour l'addition si on introduit les nombres « 2147483647 » et « 1 » ? Expliquer.

Exercice 5

a) Tracer l'exécution du programme ci-dessous (pour chaque point d'observation, noter sur un tableau la valeur de chaque variable). À la fin de l'exécution (point d'observation 3), quel sera la value de la variable « fahr » si l'utilisateur fourni la valeur « 10 » ? Vérifier en tapant et en exécutant le programme.

b) Quel est l'effet du « %f » dans le « printf » à la ligne 22 ? Quel est la différence par rapport au « %d » utilisé précédemment ?

```
 2  #include <stdio.h>
 3
 4  /* Conversion C-F */
 5  main() {
 6    float celc, fahr;
 7
 8    fahr = 0.0;
 9    celc = 0.0;
10
11    //point d'observation 1
12
13    printf("Temperature (C): ");
14    scanf("%f",&celc);
15
16    //point d'observation 2
17
18    fahr = (celc*9)/5 + 32;
19
20    //point d'observation 3
21
22    printf ("Temperature (F): %f\n",fahr);
23
24  }
25
```

Exercice 6

Tracer l'exécution du programme ci-dessous. Que fait ce programme ? Vérifier en le tapant et en l'exécutant.

```
#include<stdio.h>

/* Suite de Fibonacci de 1 a 7
 * La suite de Fibonacci est un probleme mathematique defini comme suit:
 *    f0 = f1 = 1
 *    fn+2 = fn + fn+1, pour tout n >= 0
 */

main()
{
    /* variables:
     * - p comme precedent
     * - s comme suivant
     */
    int p, s, n;

//Point d'observation 1
    printf("%i\n", n);

    p=1;
    s=1;

//Point d'observation 2

    n=s;
    s=p+s;
    p=n;    //Point d'obervation 3
    n=s;
    s=p+s;
    p=n;    //Point d'obervation 4
    n=s;
    s=p+s;
    p=n;    //Point d'obervation 5
    n=s;
    s=p+s;
    p=n;    //Point d'obervation 6
    n=s;
    s=p+s;
    p=n;    //Point d'obervation 7

    printf("Le septieme terme de la suite de Fibonacci vaut %i\n", s);
}
```

Exercice 7

a) On souhaite créer un programme qui fait l'échange entre les valeurs de deux variables (la variable « a » reçoit la valeur de la variable « b » et vice-versa). Le programme « echange1.c » illustré ci-dessous résout-il ce problème ? Si non, comment pouvons-nous résoudre cette question ?

```c
#include<stdio.h>
main()
{
    int a, b;
    a=3;
    b=5; //Point d'observation 1
    printf("a vaut %i\n", a);
    printf("b vaut %i\n", b);
    a=b;
    b=a; //Point d'observation 2
    printf("a vaut %i\n", a);
    printf("b vaut %i\n", b);
}
```

b) Pour résoudre le problème posé par la question 7a, proposer un programme pour échanger les valeurs des variables « a » et « b » à l'aide d'une troisième variable « t », dite variable « tampon ».

Informatique S1 – Programmation C
Exercices

TD 4 : Les boucles *while* et *do while*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des boucles *while* et *do while* dans le langage C.

Exercice 1

a) Exécuter le programme ci-dessous. Que fait-il ? Combien de fois l'expression « `count = count + 1 ;` » (ligne 16) sera exécutée (minimum / maximum) ?

b) Réécrire le programme ci-dessous en utilisant la boucle ***`while`*** à la place de la boucle ***`do while`***.

```
1
2   #include <stdio.h>
3
4   /*  TD 4 : boucle Do-While    */
5
6   int main () {
7       int n = 0;
8       int count = 0;
9
10      do {
11          printf ("Donner un entier > 0 : ");
12          scanf ("%i", &n);
13
14          printf ("Vous avez fourni %i \n", n);
15
16          count = count + 1;
17      } while (n > 0);
18
19      printf ("Vous avez entre %d n. entiers positifs \n", count);
20  }
21
```

Exercice 2

Comparer les deux programmes (code A et code B) ci-dessous lorsque l'utilisateur fourni le chiffre « 1 » à la première demande et lorsque l'utilisateur fourni le chiffre « 0 » à la première demande. Expliquer.

Code A :

```c
#include <stdio.h>

/*  TD 4 : Calculer la somme des numeros entres par l'utilisateur
           version avec while
*/

int main ()
{
    int n = 0;
    int count = 0;
    int sum = 0;

    /* lecture du premier nombre */
    printf ("Entrer un entier (0 pour terminer) : ");
    scanf ("%d",&n);

    while (n != 0)  {
        sum = sum + n;
        count = count + 1;

        /* lecture d'un nouveau nombre */
        printf ("Entrer un entier (0 pour terminer) : ");
        scanf ("%d",&n);
    }

    printf ("La somme des %d entiers fournis est : %d \n", count, sum);

}
```

Code B :

```c
#include <stdio.h>

/*  TD 4 : Calculer la somme des numeros entres par l'utilisateur
           version avec do-while
*/

int main ()
{
    int n = 0;
    int count = 0;
    int sum = 0;

    /* lecture du premier nombre */
    printf ("Entrer un entier (0 pour terminer) : ");
    scanf ("%d",&n);

    do  {
        sum = sum + n;
        count = count + 1;

        /* lecture d'un nouveau nombre */
        printf ("Entrer un entier (0 pour terminer) : ");
        scanf ("%d",&n);
    } while (n != 0);

    printf ("La somme des %d entiers fournis est : %d \n", count, sum);

}
```

Exercice 3

a) Remplir les trous dans le programme ci-dessous, sachant qu'il doit calculer x^y, avec x et y sont fournis par l'utilisateur.

```
#include <stdio.h>

int main () {
   float x, y; /* valeur fournis par l'utilisateur */
   float __;    /* z = x puissance y */
   int i;

   /* lecture des variables */
   ______ ("Entrez x : ");
   scanf ("__", &x);
   printf ("Entrez y : ");
   scanf ("%f", __);

   z = 1;
   i = 1;

   /* on multiplie x y-fois */
   while (_______) {
        z = z * x;
        i = _____;           /* on augmente le compteur */
   }

   /* presentation des resultats */
   printf ("x ^ y = %f \n", ___);
}
```

b) *Question « défi »* : Pouvez-vous proposer un second programme qui fait la même chose sans utiliser une boucle ? Comment ?

Exercice 4

a) Sachant que le factoriel d'un numéro entier *n* est égale à *1 * 2 * … * n-1 * n*, remplir les trous du programme ci-dessous, lequel calcule le factoriel de *n*, avec *n* fourni par l'utilisateur.

```
#include <stdio.h>

/*  TD 4 : Factoriel de n
 *  Factoriel de n = 1 * 2 * ... * n-1 * n
 */

int main () {
  int n;
  int i ___;
  int fact ____;  /* factoriel de n */

  printf ("Entrez n : ");
  ______ ("%d", &n);

   do {
       fact = fact * ____;
       _________;
  } while ( i <= n );

  printf ("Factoriel de %d est %d \n", n, fact);
}
```

b) Réécrire le programme ci-dessus pour qu'il utilise l'instruction « `while` » à la place de « `do … while` ».

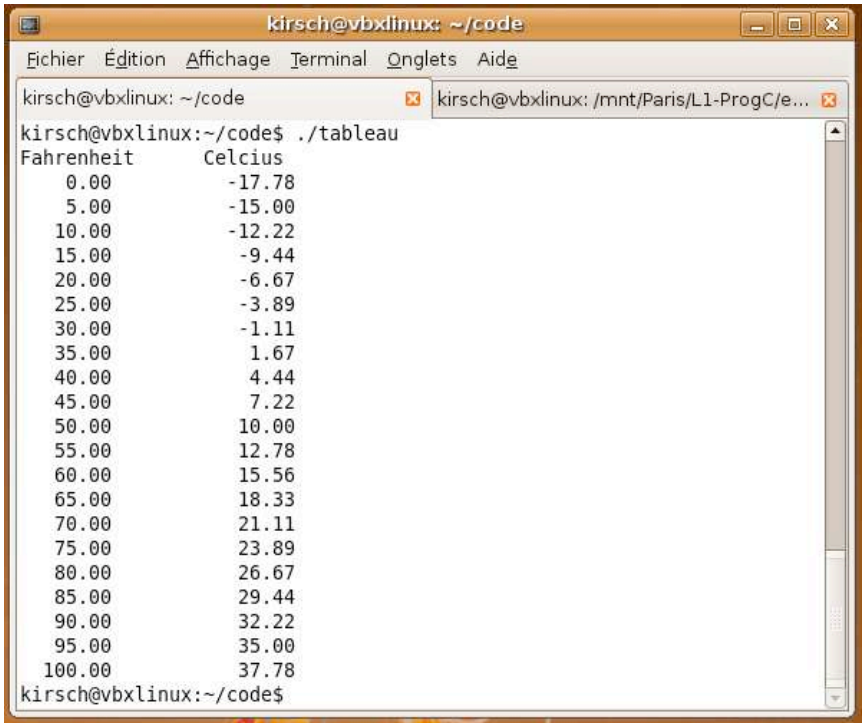
Exercice 5

Sachant que la moyenne d'un ensemble de *n* numéros réels est définie comme la somme de ces numéros divisée par *n* ($moyenne = {\sum_{i=1}^{n} x_i}/{n}$), calculer la moyenne entre 10 numéros réels fournies par l'utilisateur. (Astuce : partir de l'exercice 2).

Exercice 6

Ecrire un programme qui affiche un tableau pour convertir les degrés Fahrenheit en degrés Celsius. Par exemple, votre programme affichera sur la colonne de gauche les nombre de 0 à 100 (de 5 en 5) correspondant aux degrés Fahrenheit, et sur la colonne de droite les valeurs correspondantes en Celcius (voir la figure ci-dessous).

Rappel : $c = 5 * {(f - 32)}/{9}$



Exercice 7

a) Observer le code ci-dessous. Est-il correct ? Que fait-il ? Comment faire pour qu'il soit compréhensible ?

```
#include <stdio.h>
#include <math.h>
int main () {int i; float
x, xpow; float racx;
printf ("Donnez un nb : ");
scanf ("%f",&x); i=0; xpow=1;
while (i<x) {xpow=xpow*x; i=i+1;}
racx=sqrt(x); printf ("%f ^ %f = %f \n sqrt = %f\n",
x, x, xpow, racx);
return (0);
}
```

b) Tracer le programme ci-dessous (en considérant que l'utilisateur a fourni la valeur 4). Est-il équivalent au programme ci-dessus (question 7a) ?

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

int main ()
{
    int i;
    float x, xpow, racx;

    /* point d'observation 1 */
    printf ("Donnez un nb : ");
    scanf ("%f",&x);

    i=0;
    xpow=1;

    /* point d'observation 2 */
    while (i<x)
    {
        xpow=xpow*x;
        i=i+1;
        /* point d'observation 3 */
    }

    racx=sqrt(x);
    printf ("%f ^ %f = %f \n sqrt = %f\n",
            x, x, xpow, racx);

    /* point d'observation 5 */

    return (0);
}
```

NOTE : Afin de compiler sous Linux, les programmes qui utilisent la bibliothèque « *math.h* » (`#include <math.h>`), il faut utiliser l'option « -lm » du compilateur *gcc* : `gcc -lm -o programme programme.c` .

Exercice 8

Tracer le programme ci-dessous. Qu'affichera-t-il lorsque l'utilisateur lui fournit la séquence de numéros 4, 1 et 0 ?

```c
#include <stdio.h>
#include <math.h>
/*  TD 4 : Calculer les racines carres  */
int main () {
   float x, racine;

   /* point d'observation 1 */
   do {
     printf ("Entrer un nb (0 pour terminer) : ");
     scanf ("%f", &x);

     /*point d'observation 2 */
     racine = sqrt (x);

     printf ("La racine carre de %f est %f \n", x, racine);

   } while (x != 0);

   /*point d'observation 3 */
}
```

Exercice 9

Trouver l'erreur dans le code ci-dessous, sachant qu'il sert à calculer la *suite de Fibonacci* (définie ci-dessous) tant que l'utilisateur accepte de continuer en répondant avec un « 0 » (zéro) à la question posée :

Série de Fibonacci : $u_1 = 1$

$u_2 = 1$

$u_n = u_{n-1} + u_{n-2}$ pour $n \geq 2$

```
#include <stdio.h>

/*  TD 4 : Suite de Fibonacci
 *  La suite de Fibonacci est un probleme mathematique defini comme suit:
 *    u0 = u1 = 1
 *    un+2 = un + un+1, pour tout n >= 0
 */

int main () {
 int u0, u1;
 int un;
 int op;
 int i;


 u0 = 1;    /* u0 = 1 */
 u1 = 1;   /* u1 = 1 */
 i = 2;

 printf ("Serie de Fibonacci \n u0 = 1 \n u1 = 1 \n");

 do {
   /* un = un-1 + un-2 */
   un = u0 + u1;

   printf (" u%d = %d\n", i, un);

   u0 = u1;    //on garde les 2 dernier valeurs
   u1 = un;

   printf ("Voulez-vous continuer ( 0 = oui, 1 = non) : ");
   scanf ("%d", &op);
 } while (op = 0);

}
```

Exercice 10 (Avancé) : Approximation de Pi

Dans la mathématique, des nombreuses suites ou séries convergent vers π ou vers un multiple de π. Ces séries sont souvent à l'origine de calculs de valeurs approchées de ce nombre. Parmi ceux-là se trouve la *Fonction zêta de Riemann $\zeta(2)$*, laquelle peut être définie comme suit :

$$\zeta(2) = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{k^2} + \cdots = \frac{\pi^2}{6}$$

Réaliser un programme qui calcule cette série jusqu'un numéro *k* fourni par l'utilisateur et qui compare le résultat obtenu à la valeur de $\pi^2 / 6$.

Vous pouvez utiliser comme base le programme ci-dessous, qui affiche la valeur de π et de π^2:

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

int main () {

 double pi_carre = 0.0;
 double pi = M_PI;    /* valeur de PI selon la bibliotheque math.h */

 /* pi au carre */
 pi_carre = pow (pi, 2);

 printf ("PI = %lf \t PI^2 = %lf \n", pi, pi_carre);

 return (0);
}

```

Informatique S1 – Programmation C
Exercices

TD 5 : Les boucles *for*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des boucles *for* et l'usage des boucles imbriquées dans le langage C.

Exercice 1

Remplir les trous dans le programme ci-dessous, sachant qu'il doit calculer x^y, avec `x` et `y` sont fournis par l'utilisateur.

```
#include <stdio.h>

int main () {
   _____ x, y;  /* valeur fournis par l'utilisateur */
   float ___;
   int   i;

   /* lecture des variables */
   ______ ("Entrez x : ");
   scanf  ("%f", &x);
   printf ("Entrez y : ");
   ______ ("%f", &y);

   /* on multiplie x y-fois */
   for (____ , _____; i < y; _____ ) {
        z = z * x;
   }

   /* presentation des resultats */
   printf ("x ^ y = %f \n", z);
}
```

Exercice 2

Sachant que le factoriel d'un numéro entier *n* est égale à *1 * 2 * … * n-1 * n*, écrire un programme qui calcule le factoriel d'un numéro *n* donné par l'utilisateur, en utilisant pour cela la boucle ***for***.

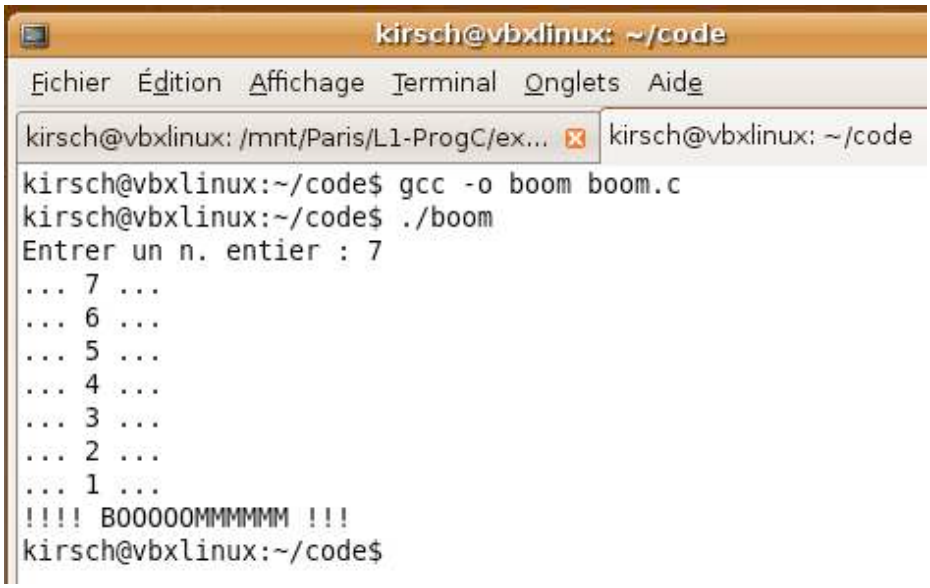
Exercice 3

Utiliser la boucle ***for*** pour calculer la moyenne entre 10 numéros réels fournis par l'utilisateur.

Rappel : la moyenne d'un ensemble de *n* numéros est la somme de ces numéros divisée par *n*

Exercice 4

Ecrire un programme qui réalise un compte à rebours à partir d'un nombre *n* fourni par l'utilisateur, tel que celui présenté dans la figure ci-dessous.



Exercice 5

a) Ecrire un programme qui utilise une boucle ***for*** pour afficher un tableau qui affiche la conversion des degrés Celsius (de -15°C à 45°C) en degrés Fahrenheit.

Rappel : $c = 5 * {}^{(f-32)}/_{9}$

b) Modifier le programme précédent pour qu'il affiche les degrés de 5 en 5 (-15, -10, -5, …, 35, 40, 45).

```
Celsius Fahrenheit
 -15.0    5.00
 -10.0    14.00
 -5.0     23.00
 0.0      32.00
 5.0      41.00
 10.0     50.00
 15.0     59.00
 20.0     68.00
 25.0     77.00
 30.0     86.00
 35.0     95.00
 40.0     104.00
 45.0     113.00
```

Exercice 6

Pour chacun des deux programmes suivants, les recopier en les indentant, faire leur tracer, et dire ce qui s'affiche à l'exécution. Expliquer la différence entre les deux programmes, et dire lequel des deux correspond le mieux aux spécifications annoncées dans les commentaires.

Code A :

```
/* Programme 1 */
/* Affichage des carre et des cube des petits entiers */
#include<stdio.h>
main(){
int a, carre, cube;
printf("Nombre\t Carre\t Cube\n");
//Point d'observation 1
for(a=1; a<=5; a++){
carre = a*a;
cube = carre*a;
//Point d'observation 2
printf(" %i\t", a);
printf(" %i\t", carre);
printf(" %i\n", cube);
//Point d'observation 3
}
}
```

Code B :

```
/* Programme 2 */
/* Affichage des carre et des cube des petits entiers */
#include<stdio.h>
main(){
int a, carre, cube;
printf("Nombre\t Carre\t Cube\n");
//Point d'observation 1
for(a=1; a<=5; a++)
carre = a*a;
cube = carre*a;
//Point d'observation 2
printf(" %i\t", a);
printf(" %i\t", carre);
printf(" %i\n", cube);
//Point d'observation 3
}
```

Exercice 7

a) Remplir les trous du programme ci-dessous. Que fait-il ?

```
#include <stdio.h>

int main () {
  int i, j ;
  int max = 10;

  /* affiche l'entete */
  printf ("      #");
  _____ (i=1; i<=max; i++)
     printf ("%4d", i);
```

```
  printf ("\n");
  printf ("########");

  for (_____; i<max; i++)
     printf ("####");

  printf ("\n");


  /* affiche les differents lignes */
  for (i=1; ______; i++) {
     printf ("%4d  #", i);
     for (_____; j<=max; _____) {
         printf ("%4d", i*j);
     }
     printf ("\n");
  }
}
```

b) Comment faire en sorte que le tableau affiche les mêmes calculs jusqu'au numéro 15 ?

Exercice 8

a) Ecrire un programme qui demande 2 entiers *x*, *y* à l'utilisateur et qui affiche en retour un rectangle de x lignes et y colonnes. Par exemple, si l'utilisateur choisit x = 5, y = 3 le programme devra afficher :

```
xxx
xxx
xxx
xxx
xxx
```

b) Modifier le programme précédent. Cette fois, un seul nombre x est demandé et le programme affiche un triangle de x lignes comme suit :

```
x
xx
xxx
xxxx
xxxxx
```

c) Dernière modification, le programme doit afficher le triangle ci-dessous pour x = 5 :

```
    x
   xxx
  xxxxx
 xxxxxxx
xxxxxxxxx
```

Informatique S1 – Programmation C
Exercices

TD 6 : Les tests *if*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des tests *if* et l'usage des tests emboités dans le langage C.

Exercice 1

a) Remplir les trous du code suivant. Que fait ce programme ?

```
#include <_________>

/*  TD 6 : lecture de 3 nb entiers, identification
 *         du plus petit et du plus grand  */

int main () {
   int ___ , ____;

   /* lecture des variables */
   printf ("Entrer a : ");
   scanf ("%d", &a);

   printf ("Entrer b : ");
   scanf ("%d", &b);

   if ( ______ ) {
      printf (" a < b \n");
   }
   ______ {
      printf (" a >= b \n");
   }

}
```

b) Quel message sera affiché si l'utilisateur entre deux numéros égaux (3 et 3, par exemple) ?

c) Etendre le programme ci-dessus pour qu'il affiche le message « a == b » si les nombres fournis par l'utilisateur sont égaux.

Exercice 2

Créer un programme qui lit trois numéros entiers et affiche à l'écran le plus petit numéro fourni par l'utilisateur. Par exemple, si l'utilisateur fourni les nombres 3, 5 et 1, le programme doit afficher « le nombre le plus petit est 1 ».

Exercice 3

Ecrire un programme qui demande à l'utilisateur deux entiers a et b et qui indique en retour si b divise ou non a.

Exercice 4

Ecrire un programme qui demande un numéro entier à l'utilisateur et qui lui répond si le numéro est pair ou impair.

Exercice 5

Ecrire un programme qui lit une note *n* (sur 20), puis affiche la mention correspondante : « ajourné » si $n < 10$, « passable » si $10 \leq n < 12$, « assez bien » si $12 \leq n < 14$, « bien » si $14 \leq n < 16$, et « très bien » sinon.

Exercice 6 - Avancé

Les diviseurs d'un nombre entier *n* sont tous les numéros entiers positifs *i* inférieurs à *n* ($0 \leq i \leq n$) qui divisent *n*. Etant donné un numéro entier *n* fourni par l'utilisateur, afficher tous les diviseurs de *n*.

Exercice 7 - Avancé

a) Calculer la moyenne d'un ensemble de notes fournies par l'utilisateur. L'utilisateur peut fournir autant des notes qu'il veut. Pour terminer le programme et afficher la moyenne, l'utilisateur doit fournir un numéro négatif lequel ne doit pas être pris en compte dans le calcul de la moyenne.

b) Etendre le programme précédent pour qu'il affiche également si cette moyenne est « insatisfaisante » si la moyenne est inférieur à 10, « passable » si elle est entre 10 et 12 (compris), « assez bien » si elle est entre 12 et 14, « bien » si la moyenne est entre 14 (compris) et 16, et « très bien » sinon.

Informatique S1 – Programmation C
Exercices

TD 7 : *switch*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage de l'instruction *switch* en langage C.

Exercice 1

Remplir les trous du code suivant. Que fait ce programme ?

```
#include <stdio.h>

/*  TD 7 : choix de menu */

int main () {
    int choix;
    /* affichage du menu */
    printf ("Indiquer votre preference :\n");
    printf ("I============================I\n");
    printf ("I [1] - Menu salade           I\n");
    printf ("I [2] - Menu sand. + salade   I\n");
    printf ("I [3] - Menu burger           I\n");
    printf ("I [4] - Menu burger frites    I\n");
    printf ("I [5] - Aucun des precedents  I\n");
    printf ("I============================I\n");
    printf ("\n");

    /* demande le choix */
    printf ("Choix : ");
    scanf ("%d",&choix);

    _________ (choix) {
        case 1:
                printf ("Choix : Menu salade \n");
                printf ("        Bravo ! Votre sante vous remercie.\n");
                _______;
        case 2:
                printf ("Choix : Menu sans + salade \n");
                printf ("        C'est bien de faire attention a votre
ligne.\n");
                break;
        ______ 3:
                printf ("Choix : Menu burger \n");
                printf ("        Attention aux repas non-equilibres\n");
                break;
        case 4:
                printf ("Choix : Menu burger frites \n");
                printf ("        Attention a votre cholesterol.\n");
                break;
        case ___:
                printf ("Choix : Aucun des precedents \n");
                printf ("    N'oubliez pas qu'il faut manger
equilibre.\n");
                break;
         _________:
                printf ("Choix invalide! \n");
    }
}
```

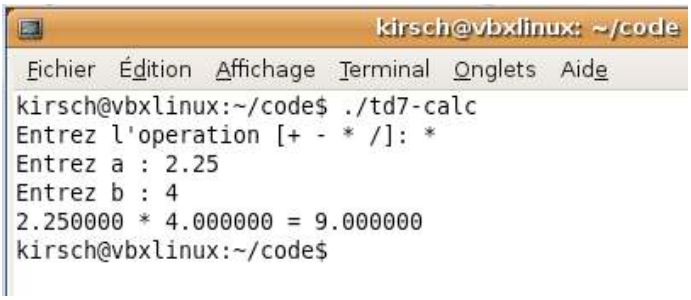
Exercice 2

Ecrire un programme qui demande à l'utilisateur d'entrer un caractère et qui affiche en retour si le caractère entré est une voyelle (a, e, i, o, u), un chiffre (0, 1, 2, …, 9), une consonne (b, c, d, f, g, h, … , t, v, x, z), ou autre (caractère de contrôle).

Astuce : voir exemple vu en cours.

Exercice 3

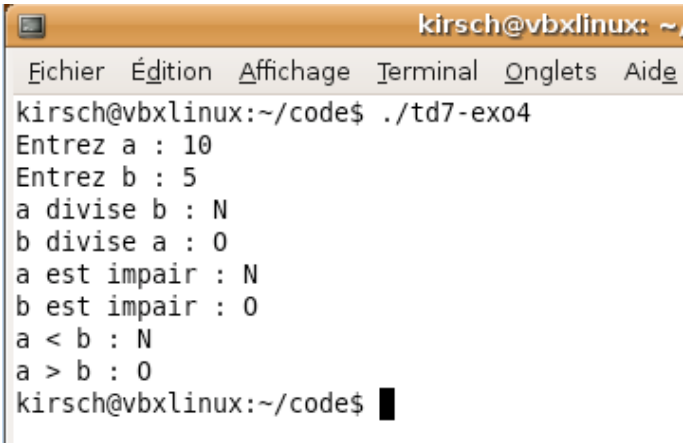
Ecrire un programme qui réalise une calculette simple. L'utilisateur doit fournir une opération (`+`, `-`, `*`, `/`) et deux numéros réels. A l'aide de l'instruction `switch`, le programme doit réaliser l'opération souhaitée et afficher le résultat à l'utilisateur.



Exercice 4

Réaliser un programme qui demande à l'utilisateur deux nombres entiers `a` et `b`, et qui à l'aide de l'opérateur « `?` **:** » affiche les réponses ('O' ou 'N') aux questions suivantes (voir copie d'écran ci-dessous) :

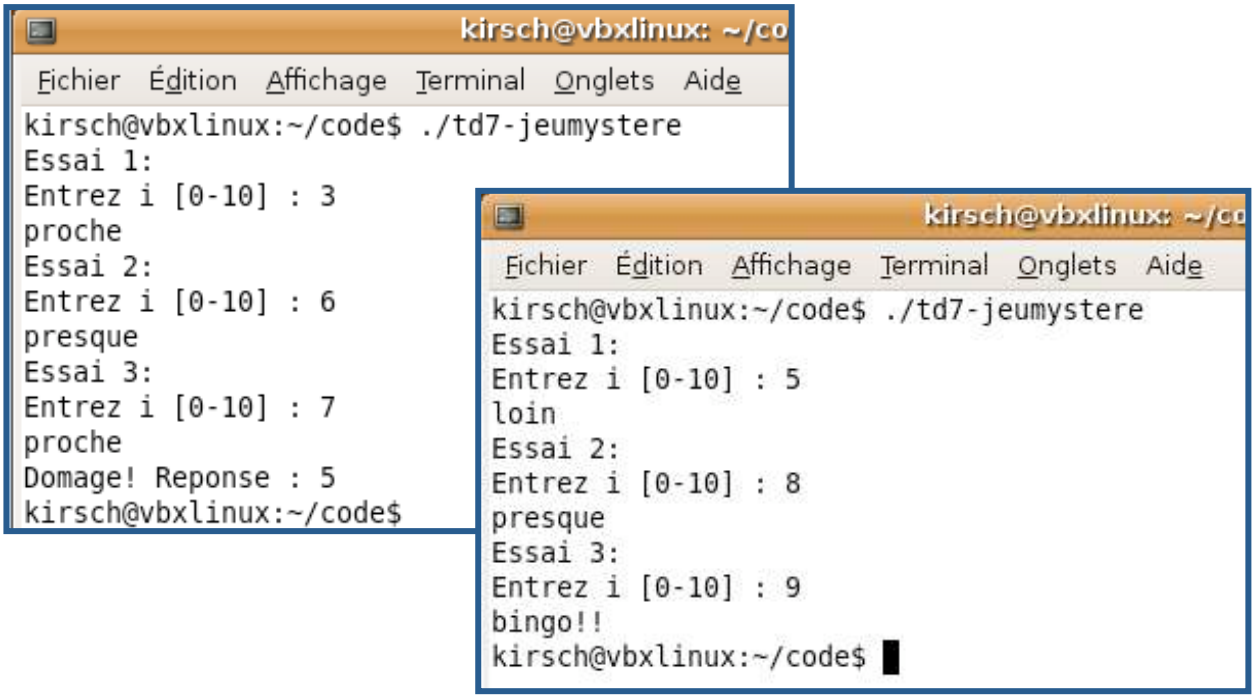
a) a divise b ?
b) b divise a ?
c) a est impair ?
d) b est impair ?
e) a est plus petit que b ?
f) b est plus petit que a ?



Exercice - Avancé

Jeu du numéro mystère : l'utilisateur doit trouver un numéro entre 0 et 10 calculé de manière aléatoire par le programme. L'utilisateur a droit à trois essais pour trouver le bon numéro. S'il échoue après les trois essais, le programme doit afficher le numéro mystère. Après chaque essai, le programme doit informer à l'utilisateur s'il a trouvé la bonne réponse, si le numéro fourni est proche du numéro mystère, ou si, au contraire, il est loin du numéro mystère.

Astuce : étendre l'exemple présenté en cours.



Informatique S1 – Programmation C
Exercices

TD 8 : codage binaire et caractères ASCII

Dans ce TD, nous allons réaliser des exercices couvrant la représentation des caractères en langage C.

Avant de réaliser les exercices ci-dessous, changer dans le terminal le codage des caractères vers « Occidental » (menu Terminal → Définir le codage des caractères → Occidental (ISO-8859-15).

Exercice 1

a) Ecrire un programme qui demande à l'utilisateur un caractère et qui affiche en retour le code ASCII du caractère fourni.

Exercice 2

a) Ecrire un programme qui affiche les caractères ASCII de 0 à 255.

Astuce : utiliser une boucle *for* pour parcourir tout l'intervalle.

b) Améliorer le programme précédent de manière à ce qu'il affiche l'ensemble des caractères ASCII dans un tableau de 10 colonnes séparées par un ' | '.

			!	"	#	$	%	&	'	(
)	*	+	,	-	.	/	0	1	2	
3	4	5	6	7	8	9	:	;	<	
=	>	?	@	A	B	C	D	E	F	
G	H	I	J	K	L	M	N	O	P	
Q	R	S	T	U	V	W	X	Y	Z	
[	\	]	^	_	`	a	b	c	d	
e	f	g	h	i	j	k	l	m	n	
o	p	q	r	s	t	u	v	w	x	
y	z	{	|	}	~					
¡	¢	£	€	¥	Š	§	š	©	ª	
«	¬	-	®	¯	°	±	²	³	Ž	
µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	
¿	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	
É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	
Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	
Ý	Þ	ß	à	á	â	ã	ä	å	æ	
ç	è	é	ê	ë	ì	í	î	ï	ð	
ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	

Exercice 3

a) Ecrire un programme qui dessine un arbre de Nöel avec le caractère ASCII n° 174.

Astuce : observer le code de l'exercice 8 de la TD n° 5 pour imprimer l'arbre de Nöel.

```
kirsch@vbxlinux:~/code$ ./arbrenoel
          ®
         ®®®
        ®®®®®
       ®®®®®®®
      ®®®®®®®®®
     ®®®®®®®®®®®
    ®®®®®®®®®®®®®
   ®®®®®®®®®®®®®®®
  ®®®®®®®®®®®®®®®®®
 ®®®®®®®®®®®®®®®®®®®
®®®®®®®®®®®®®®®®®®®®®
kirsch@vbxlinux:~/code$
```

b) Etendre le programme précédent pour qu'il affiche un arbre de Nöel (toujours avec le caractère ASCII n° 174) comme celle affichée dans l'image ci-dessous.

```
kirsch@vbxlinux:~/code$ ./arbrenoelb
          ®
         ®®®
        ®®®®®
       ®®®®®®®
      ®®®®®®®®®
     ®®®®®®®®®®®
    ®®®®®®®®®®®®®
   ®®®®®®®®®®®®®®®
  ®®®®®®®®®®®®®®®®®
 ®®®®®®®®®®®®®®®®®®®
®®®®®®®®®®®®®®®®®®®®®
         ®®®®
         ®®®®
         ®®®®
         ®®®®
kirsch@vbxlinux:~/code$
```

Exercice 4 – Avancé

a) Etendre le programme de l'exercice 1 de manière à ce que l'utilisateur puisse répéter l'opération tant qu'il le souhaite (voir écran ci-dessous).

Note : pour chaque caractère donné par l'utilisateur, en réalité, deux caractères sont fournis : celui donné par l'utilisateur et le caractère de nouvelle ligne (la touche « entrée »).

```
kirsch@vbxlinux:~/code$ ./td8-exo4
Entrez un caractere : A
Code ASCII de A est 65
Continuer [ O | N ] ? O
Entrez un caractere : Z
Code ASCII de Z est 90
Continuer [ O | N ] ? O
Entrez un caractere : a
Code ASCII de a est 97
Continuer [ O | N ] ? O
Entrez un caractere : z
Code ASCII de z est 122
Continuer [ O | N ] ? n
kirsch@vbxlinux:~/code$
```

b) Compter le nombre de lettres majuscules, minuscules et des chiffres fournis par l'utilisateur dans le programme précédent.

```
kirsch@vbxlinux:~/code$ ./td8-exo4b
Entrez un caractere : 0
Code ASCII de 0 est 48
Continuer [ O | N ] ? O
Entrez un caractere : 9
Code ASCII de 9 est 57
Continuer [ O | N ] ? O
Entrez un caractere : A
Code ASCII de A est 65
Continuer [ O | N ] ? O
Entrez un caractere : a
Code ASCII de a est 97
Continuer [ O | N ] ? n
Vous avez entrer : 2 chiffres,
         1 lettres majuscules
         1 lettres minuscules
kirsch@vbxlinux:~/code$
```

Informatique S1 – Programmation C
Exercices

TD 9-10 : fonctions et arrays

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des fonctions appartenant aux bibliothèques standards et l'usage des arrays unidimensionnels en langage C.

Exercice 1

Ecrire un programme capable de lire 15 caractères fournis par l'utilisateur et de :

- calculer combien de lettres majuscules ont été fournies
- calculer combien de lettres minuscules ont été fournies
- calculer combien de chiffres ont été fournis
- afficher les caractères fournis par l'utilisateur en majuscule

Astuces :

- utiliser les fonctions de la bibliothèque ctype.h
- chaque entrée correspond au caractère entrée et un espace (la nouvelle ligne)

Exercice 2

Ecrire un programme qui lit 10 numéros fournis par l'utilisateur et qui :

a) calcule la moyenne entre les 10 numéros fournis ;
b) affiche les extrémités (le numéro le plus petit et le plus grand) parmi les numéros fournis par l'utilisateur ;
c) affiche l'écart entre les extrémités et entre ces extrémités et la moyenne ;
d) calculer l'écart moyen entre les numéros fournis et la moyenne.

Astuce : utiliser `#define` pour définir la taille maximale du vecteur.

Exercice 3

Ecrire un programme qui demande à l'utilisateur 10 numéros réels et qui calcule la fréquence (nombre d'occurrence) de chaque numéro fourni.

Guide de référence Rapide - Linux

Commandes

ls
`ls *.c`
`ls –l nomfichier.c`
	affiche la liste des fichiers d'un répertoire
	affiche la liste des fichiers dont l'extension est « .c »
	afficher les détails (nom, taille, date…) sur le ficher « nomfichier.c »

mkdir *mkdir tds*	crée un nouveau répertoire *crée un nouveau répertoire nommé « tds »*
cd *cd ..* *cd ./tds*	change le répertoire courant *change vers le répertoire au dessus* *change vers le sous-répertoire « tds »*
rmdir `rmdir tds`	efface un répertoire vide *efface le répertoire nommé « tds »*
pwd	indique le répertoire courant (Où suis-je ?)
rm *rm nomfichier.c* *rm *.c*	efface un fichier *efface le fichier nommé « nomfichier.c »* *efface tous les fichiers dont l'extension est « .c »*
cp `cp a.c /media/usb`	copie un fichier *copie le fichier « a.c » vers le fichier « a.c » localisé dans le répertoire « /media/usb »*
mv `mv a.c /media/usb`	déplace un fichier vers une nouvelle localisation *déplace le fichier nommé « a.c » vers le répertoire « /media/usb »*

Trucs & Astuces

?	correspond à un seul caractère
*	correspond à 0 ou plus caractères
.	désigne le répertoire courant
..	désigne le répertoire juste au dessus du répertoire courant dans l'arborescence
/dir/subdir/fichier **./fichier** **../subdir/fichier**	chemin complet d'accès à un fichier chemin relatif d'accès à un ficher localisé dans le répertoire courant
| `cat a.txt | more`	permet de concaténer (enchainer) la sortie d'une commande à l'entrée de la suivante
>`cat a.c > a.txt`	(redirection) permet de rediriger la sortie d'une commande vers un fichier
>>`cat b.c >> a.txt`	(*append*) permet d'annexer la sortie d'une commande dans un autre fichier
<`less < a.c`	(redirection) permet de rediriger un fichier vers l'entrée d'une commande

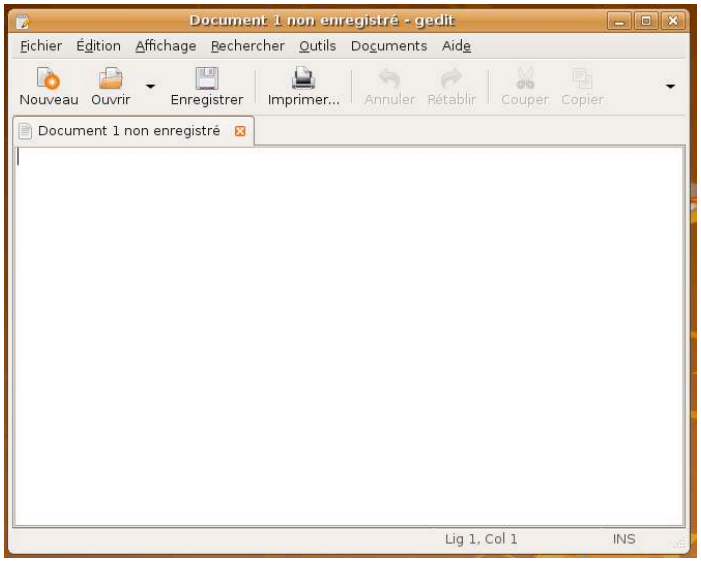
\
`cat \`
`a.c`
	permet de continuer une commande sur deux lignes

Université Paris 1 – Panthéon Sorbonne
Manuele Kirsch Pinheiro

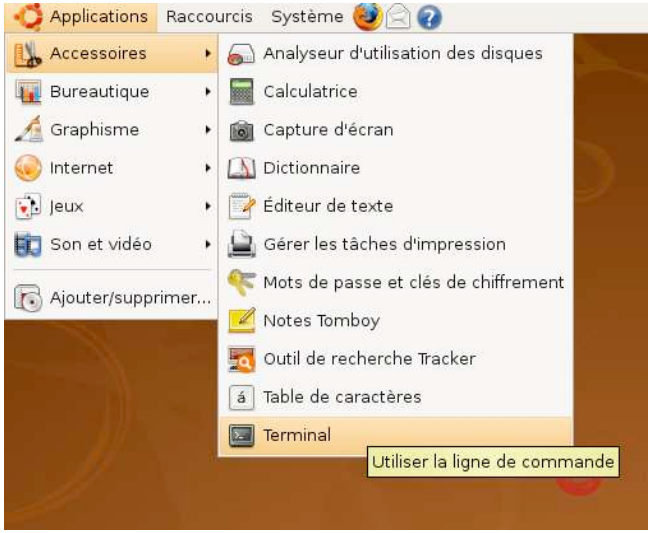
cat ***cat a.c***	présente le contenu d'un fichier *imprime le contenu du fichier nommé « a.c »*	↑	permet de rappeler la dernière commande tapée
more ***more a.c***	imprime le contenu d'un fichier page par page	**TAB**	permet de compléter automatiquement le nom d'un fichier ou d'une commande
whoami	retourne le nom (login) de l'utilisateur	**CTRL-D**	EOF (*end of file*)
less ***less a.c***	imprime le contenu d'un fichier page par page (idem more)	**CTRL-C**	permet d'interrompre l'exécution d'un programme
clear	efface l'écran	**Commandes Avancées**	
which **which gcc**	retourne le chemin complet d'accès à une commande	**grep** *grep "main" *.c*	cherche une expression (mot) dans un fichier *cherche le mot « main » dans tous les fichiers « .c »*
whereis **whereis gcc**	retourne le chemin complet d'accès à tous les répertoires contenant la commande	**mount** mount /mnt/removable mount /dev/sdb \ /mnt/removable	connecte un *filesystem* (*e.g.* dispositif) à un répertoire donné
man ***man grep***	présente l'aide pour une commande *présente l'aide sur la commande « grep »*	**umount** mount /mnt/removable	déconnecte un *filesystem*
zip **zip codes.zip *.c**	compresse (archive) un fichier (crée un fichier .zip)	**ps**	liste les processus (programmes) qui exécutent
unzip **unzip codes.zip** **unzip –l codes.zip**	décompresse un ficher compressé avec zip (.zip) liste le contenu du fichier « codes.zip »	**kill** ***kill 1196***	termine un processus (programme) identifié par son numéro (PID), visible avec *ps*
gzip **gzip file.txt**	compresse (archive) un fichier (crée un fichier .gz)	**echo** ***echo "Hello word"*** ***echo $PATH***	répète la phrase qui lui a été donné (echo) *répète la phrase « Hello world »* *montre le contenu de la variable « PATH » (chemin vers les programmes)*
gunzip **gunzip file.txt.gz**	décompresse un ficher compressé avec gzip (.gz)	**exit**	quitter

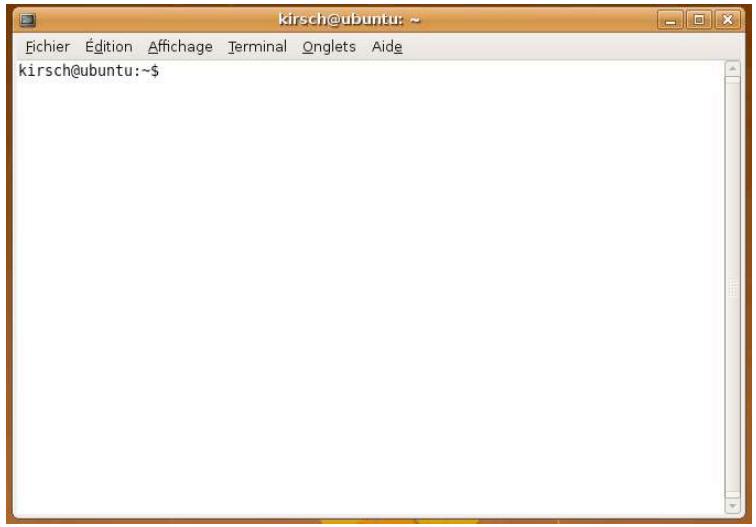
Environnement graphique Gnome

Sous Linux, on utilise souvent l'environnement graphique ***Gnome*** (important : il y a d'autres environnements graphiques sous Linux). Celui-ci propose plusieurs applications particulièrement simples et utiles. Parmi ces applications, deux méritent notre attention ici, puisqu'elles peuvent être utilisées pour la programmation en C : l'éditeur de texte ***gedit*** et le ***terminal***. La première permet l'édition facile des documents texte (non-formatés), tandis que la deuxième donne accès à la ligne de commande.



Université Paris 1 – Panthéon Sorbonne
Manuele Kirsch Pinheiro





Sur Windows

Plusieurs compilateurs C et environnements de programmation (IDE) sont disponibles sur MS Windows©. Parmi ceux-ci, nous pouvons souligner :

- MinGW - Minimalist GNU for Windows (compilateur) : http://www.mingw.org/
- Code::Blocks (IDE) : http://www.codeblocks.org/home
- Dev-C++ (IDE) : http://www.bloodshed.net/dev/devcpp.html
- Notepad++ (éditeur texte simple) : http://notepad-plus.sourceforge.net/fr/site.htm

Sources

Nicolas Trotignon, Langage C – L1 – MASS, TDs de cours, Université Paris 1, 2007-2008.

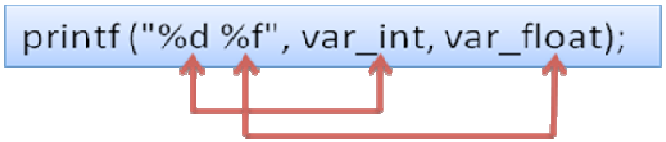
Armand Delcros, « Les commandes fondamentales de Linux ». Disponible sur : http://www.linux-france.org/article/debutant/debutant-linux.html (dernière visite : 11/09/2008)

Annie Danzart, « Abrégé Unix », École Nationale Supérieure des Télécommunications. Disponible sur : http://www.infres.enst.fr/~danzart/unix_abrege.html (dernière visite : 11/09/2008)

Arnold Robbins, « Bash Quick Reference ». Disponible sur : http://www.digilife.be/quickreferences/quickrefs.htm (dernière visite : 11/09/2008)

Printf

printf ("texte + format", variables….) ;



Formats

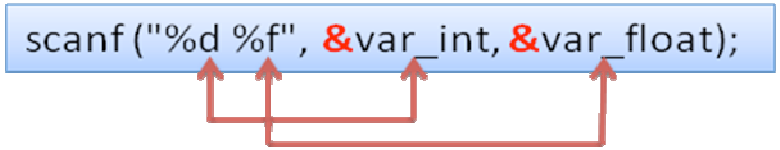
• **%d** int • **%i** int • **%f** float / double • **%e** (notation exponentielle) • **%c** char • %ld long • %s chaîne de caractères	• %*n*d int avec un minimum de *n* chiffres (ou espaces) o %3d __1 (le _ représente un espace) • %*n*f float avec un minimum de *n* caractères (les chiffres et le point) • %.*n*f float avec *n* chiffres (ou zéros) après le point o %10f ____1.2345 (le _ représente un espace) o %.2f 1.23 o %10.2f ______1.23

Exemples

```
int entier = 1;
float reel = 1.2345;        /* attention : c'est "." et non "," */
char lettre = 'a';          /* pour les caracteres, on a besoin des ' ' */
printf ("Hello World ! \n");
printf ("un entire : %d    un reel : %f      un char : %c \n", entier, reel, lettre);
printf ("un entier : %3d   un reel : %10.2f  un char : %c\n", entier, reel, lettre);
```

Scanf

scanf ("format", &variable….) ;



Formats

•	**%d**	int
•	**%i**	
•	**%f**	float
•	**%e**	
•	**%c**	char
•	%lf	double
•	%le	
•	%ld	long
•	%hd	short
•	%s	chaîne de caractères

Exemples

```
int entier ;
float reel ;
char lettre ;

scanf ("%d", &entier);
scanf ("%f", &reel);
scanf ("%c", &lettre);
```

Université Paris 1 – Panthéon Sorbonne
Manuele Kirsch Pinheiro

Caractères spéciaux

Quelques exemples de caractères spéciaux (à utiliser notamment avec *printf*) :

- \n nouvelle ligne
- \t tab
- \\ la « \ »
- \" le « " »
- %% le « % »

Exemple

```c
#include <stdio.h>

int main () {
    int entier;   /* un nombre entier */
    float reel;   /* un nombre reel */
    char lettre;  /* une lettre */

    printf ("Entrer une lettre : ");
    scanf ("%c",&lettre);
    printf ("Vous avez entre : \t %c \n", lettre);

    printf ("Entrer un nombre entier : ");
    scanf ("%d",&entier);
    printf ("Vous avez entre : \t %d \n",entier);

    printf ("Entrer un nombre reel : ");
    scanf ("%f",&reel);
    printf ("Vous avez entre : \t %f \n",reel);

}
```