NSY103
TP 3 – Introduction à la programmation en C

2016

Les TPs seront réalisés sous GNU/Linux.

Pour commencer, démarrez une machine sous GNU/Linux et connectez-vous en utilisant utilisant les login/mot de passe génériques: `licencep` / `7002n*`

Partie 1 : Introduction

a) Bonjour le Monde !

Pour commencer on se propose d'écrire un programme affichant "Bonjour␣le␣monde".

```c
#include <stdio.h>

int main() {

  /* Affiche la chaîne de caractère "Bonjour le monde" sur la
     sortie standard suivi d'un retour à la ligne */
  printf("Bonjour␣le␣monde\n");

  return 0; // valeur de retour du programme
}
```

Listing 1 – Programme C « Bonjour le Monde »

1. Ouvrez un terminal et vérifiez que vous vous trouvez bien dans votre répertoire de travail ;
2. Créez un répertoire pour ce TP nommé `tpc` et déplacez-vous dedans ;
3. Ouvrez un éditeur de texte et recopiez ce programme dans un fichier texte nommé `bonjour.c` que vous enregistrerez dans le répertoire précédemment créé. Le nom d'un fichier contenant du code C doit se terminer par l'extension `".c"` ;

Pour pouvoir exécuter ce programme, il faut ensuite le compiler. Pour compiler un programme C sous UNIX, on dispose d'un compilateur nommé « `cc` ». Sous GNU/Linux, ce compilateur fait partie de la suite de compilateur GCC (*the GNU Compiler Collection*).

4. Exécuter la commande `cc bonjour.c` ; Normalement le programme doit compiler sans erreur ;
5. Listez le contenu du répertoire `tpc` ; Un fichier `a.out` a dû être créé ;
6. Exécutez le programme `./a.out` ;

b) Structure du programme `bonjour`

Regardons maintenant un peu plus en détail la structure du programme `bonjour.c`.
Ce programme `bonjour.c` est constitué de deux parties :

— Un préambule qui contient ici uniquement une directive **#include**

— La définition de la fonction main.

Remarque (Déclaration vs. Définition)**.**

— *La* définition *d'une fonction comprend :*

  — *le nom de la fonction ;*

  — *le nom et le type de chacun des arguments de la fonction ;*

  — *le type de la valeur de retour de la fonction ;*

  — *le corps de la fonction (déclaration de variables locales ; suite d'instructions à exécuter) ;*

— *La* déclaration *d'une fonction comprend uniquement le* prototype *de la fonction qui spécifie le nom, le type des paramètres de la fonction*[1] *ainsi que le type de sa valeur de retour. Cela permet au compilateur de vérifier que l'appel de la fonction est correct. Par exemple, le prototype de la fonction main du programme* **`bonjour`** *est :*

```
1       // Déclaration de la fonction main
2       int main();
```

- **La Fonction main**

La fonction main est obligatoire dans tout programme C. C'est toujours la première fonction appelée au lancement d'un programme. On peut remarquer qu'ici cette fonction ne prend pas de paramètre et qu'elle retourne un entier (**int**) de valeur 0.

- **Les Commentaires**

La fonction main commence par 2 lignes de commentaires (ligne 5 et 6).
Les commentaires en C peuvent être donnés de deux façons différentes :

— Soit sous forme de bloc compris entre la balise ouvrante /∗ et la balise fermante ∗/; Sous cette forme, un commentaire peut s'étaler sur plusieurs lignes ;

— Soit sur une simple ligne comme à la ligne 9 : le commentaire doit commencer par // et se termine à la fin de la ligne.

- **La Fonction printf**

Après les 2 lignes de commentaire, on peut voir un appel à la fonction printf. La fonction printf prend en paramètre au minimum une chaîne de caractère appelée *chaîne de format*. printf retourne comme valeur le nombre de caractères imprimés (cette valeur de retour est ignorée dans le programme `bonjour`).

La chaîne de format permet de décrire le format des données à afficher qui peuvent être une chaîne de caractère constante comme "Bonjour␣le␣monde\n", où alors les valeurs contenues dans des variables (voir exercice b)).

1. Le prototype d'une fonction peut aussi inclure le nom de ses arguments

7. Supprimez les caractères '\' et 'n' de la chaîne "Bonjour␣le␣monde\n". Recompilez le programme et exécutez-le de nouveau. Que constatez vous ?

La séquence de caractère "\n" est une séquence d'échappement. Elle permet d'introduire un caractère spécial : ici, le saut de ligne. Le tableau 1 présente les séquences d'échappement les plus courantes :

Séquence d'échappement	Caractère correspondant
\a	bip système (alerte)
\n	nouvelle ligne ; saut de ligne
\t	tabulation
\\	affichage du caractère anti-slash (\)
\'	affichage d'une simple quote (')
\"	affichage d'une double quote (")

TABLE 1 – Séquences d'échappement usuelles

- **La directive return**

L'instruction **return** permet de spécifier la valeur de retour de la fonction main. Ici, la fonction main retourne la valeur 0 qui par convention signifie que le programme s'est exécuté sans erreur.

- **La directive #include**

La directive **#include** permet d'inclure la déclaration de la fonction printf. Sans cela, le compilateur ne connaîtrait pas le prototype de la fonction printf et ne pourrait pas vérifier si son appel est correct.

printf est une fonction de la librairie standard C. Elle est donc toujours disponible avec le compilateur.

Remarque. *On peut connaître les fichiers d'entête à inclure pour utiliser une fonction de la librairie standard en consultant la page de manuel correspondante de la section 3 :*

```
$ man 3 printf

PRINTF(3)    Linux Programmer's Manual            PRINTF(3)

NAME
   printf, fprintf, sprintf - formatted output conversion

SYNOPSIS
   #include <stdio.h>

   int printf(const char *format, ...);

⋮
```

Partie 2 : Manipulation des caractères

a) Lecture / Écriture de caractères en C

```c
#include<stdio.h>

int main() {
  char c;

  printf("Entrez un caractère : ");
  c = getchar();

  printf("Voici le caractère entré : %c\n", c);

  return 0;
}
```

Listing 2 – Lecture / Écriture de caractères en C

1. Recopiez, compilez et exécutez le programme du listing 2 ;

Le programme du listing 2 lit un caractère entré au clavier et l'affiche sur le terminal. On peut voir qu'il fait appel à 2 fonctions de la librairie C :

— la fonction printf vue précédemment ;

— la fonction getchar qui lit un caractère et renvoie sa valeur (getchar est déclarée dans le fichier `unistd.h`).

La fonction getchar est appelée à la ligne 8 et la valeur du caractère lu est stockée dans la variable c.

La variable c a été déclarée à la ligne 5 : **char** c;. Elle a pour type **char**. Le type **char** est généralement utilisé pour représenter les caractères. Sur la table 2, on retrouve les types de base en C ;

Remarque. *La variable c est déclarée dans le corps de la fonction main et ne sera donc* visible *qu'à l'intérieur de la fonction main (dans le bloc définit par '{' et '}'). On dit qu'il s'agit d'une* variable locale *(à la fonction main).*

nom	Signification
`char`	codé sur un octet, permet de représenter un jeu de caractères 8 bits
`int`	représente les entiers, sa taille dépend de l'architecture de la machine (4 octets sur une architecture 32 bits, 8 sur une 64 bits)
`float`	nombre en virgule flottante simple précision
`double`	nombre en virgule flottante double précision

TABLE 2 – Type de Base en C

À la ligne 10, la fonction printf affiche la chaîne de caractère "Voici le caractère entré : " suivie du caractère lu, puis, d'un saut à la ligne.

On peut voir que l'emplacement du caractère lu dans la chaîne de caractère à afficher est spécifié par une chaîne de conversion : "%c". "%c" sera remplacé par la valeur contenue dans c et cette valeur sera considérée comme celle d'un caractère. Il peut y avoir plusieurs chaînes de conversion dans une chaîne de format. La chaîne de format doit être suivie d'autant d'arguments qu'elle comprend de chaînes de conversion.

Dans l'exemple ci-dessous, il y a 4 chaînes de conversion et 4 arguments après la chaîne de format.

```
1  printf("Les lettres vont de %c à %c, et les chiffres vont de %c à
2 %c\n", 'a', 'z', '0', '9');
```

2. Remplacez la chaîne de conversion "%c" par "%d". Recompilez puis exécutez de nouveau le programme. Que constatez-vous ?

La chaîne de conversion "%d" signifie que le paramètre correspondant doit être considéré comme un entier. Les chaînes de conversion les plus courantes sont représentées table 3.

Chaîne	Signification
`%d`	affiche un entier en base dix
`%4d`	affiche un entier en base dix sur 4 chiffres
`%f`	affiche un flottant
`%4.2f`	affiche un flottant sur 4 chiffres dont 2 décimales
`%c`	affiche un caractère
`%s`	affiche une chaîne de caractères

TABLE 3 – Chaînes de conversion

b) Codage des Caractères

Le fait d'associer à un ensemble de caractères (chiffres, lettres, ...) une valeurs numérique représente un codage de caractères. L'un des codages le plus répandu est le codage ASCII (*American Standard Code for Information Interchange*) présenté figure 1.

3. Modifiez le programme précédent pour afficher un caractère et son code.

La fonction est_chiffre permet de tester si un caractère est un chiffre.

```
1 int est_chiffre(char c) {
2   if ( c >= '0' && c <= '9') {
3     return 1;
4   } else {
5       return 0;
6   }
7 }
```

Listing 3 – Fonction testant si un caractère est un chiffre

4. Modifier le programme précédant pour qu'il indique si le caractère lu est un chiffre. Utilisez pour cela la fonction est_chiffre. Recompilez le programme et testez le ;
5. À la ligne 2 de la fonction est_chiffre, les chiffres 0 et 9 sont donnés entre simples quotes, pourquoi ? Quelle en est la signification ?
6. Supprimez les simples quotes, recompilez le programme et testez-le. Comment expliquez-vous ces résultats ? Corrigez ensuite votre programme ;
7. Écrivez une fonction qui permet de tester si un caractère est une lettre (majuscule ou minuscule). Modifiez ensuite votre programme pour qu'il indique si le caractère lu est un chiffre ou une lettre.

c) Les tableaux de caractères

Les tableaux se définissent simplement en C. Il suffit de faire suivre le nom de la variable par des crochets pour indiquer que l'on souhaite déclarer un tableau.

```
type identifiant[taille]
```

On accède ensuite aux éléments du tableau en précisant l'indice de la case du tableau à laquelle on souhaite accéder. Les indices des cases du tableau de 0 à `taille`-1.

Par exemple, pour déclarer un tableau de 10 entiers :

```
int tab_entier[10]; // déclaration d'un tableau de 10 entiers
int a, b;           // déclaration d'un entier a et d'un entier b

a = 5;              // affectation de la valeur 5 à a
tab_entier[1] = a;  /* affectation de la valeur de a à la case
                       d'indice 1 du tableau tab_entier */
b = tab_entier[1];  /* affectation de la valeur de la case 1
                       du tableau tab_entier à b */
```

Le programme ci-dessous lit une suite de caractères, les place dans un tableau, puis affiche le contenu du tableau :

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

#define TAILLE 5

int main() {
  char tab[TAILLE];
  int i = 0;          /* indice nous permettant de compter le nombre
                         de caractères lus. On l'initialise à 0 */

  printf("Entrez %d caractères suivit de Entrée: ", TAILLE);
  while ( i < TAILLE ) {  /* lit les caractères (le nombre de
                             caractères à lire est donné par TAILLE) */
    tab[i] = getchar();
    i++;                        // incrémente la valeur de i de 1
  }

  // on affiche le contenu du tableau
  i = 0;
  while ( i < TAILLE ) {
    printf("%c", tab[i]);
    i++;
  }
  printf("\n");               // affiche un retour à la ligne

  return 0;
}
```

Listing 4 – Lecture et affichage d'une série de caractères

8. Compilez et exécutez le programme ci-dessous. Quelle structure de donnée avons-nous simulée ?
9. Remplacez les lignes 19 à 24 par printf("%s\n", tab); afin de demander à la fonction printf d'afficher le tableau tab comme s'il s'agissait d'une chaîne de caractères. Que se passe-t-il ? Pourquoi ?

```
1  #include <stdio.h>
2
3  #define TAILLE 4
4
5  int main() {
6    char tab[TAILLE];
7    int i = 0;          /* indice nous permettant de compter le
         nombre
8                           de caractères lus. On l'initialise à 0 */
9
10   printf("Entrez %d caractères suivit de Entrée: ", TAILLE);
11   while ( i < TAILLE ) {  /* lit les caractères (le nombre de
12                             caractère à lire est donné par TAILLE)
                                */
13     tab[i] = getchar();
14     i++;                        // incrémente la valeur de i de 1
15   }
16
17   //  tab[i]= 0;
18   // on affiche le contenu du tableau
19   printf("%s\n", tab);
20
21   return 0;
22 }
```

Listing 5 – Lecture et affichage d'une série de caractères

10. Augmenter la taille du tableau d'une case (**char** tab[TAILLE+1];) et rajoutez l'instruction suivante à la ligne 17 : tab[i]= 0;
11. Recompilez et retestez le programme. Il devrait maintenant fonctionner normalement.

Remarque. *Pour manipuler des chaînes de caractères correctement, il faut soit connaître le taille, soit être capable de déterminer quand elles se terminent. En C, les chaînes de caractères sont à* zéro terminal, *c'est à dire que le dernier caractère doit avoir comme valeur numérique 0 pour savoir que la chaîne de caractères est finie.*

d) Convertir une Chaîne de Caractères en Majuscule

```
1  #include <stdio.h>
2
3  char maj(char c) {
4    char m;
5    int diff = 'a' - 'A';
6
7    if (c >= 'a' && c <= 'z') {
8      m = c - diff;
9    } else {
```

```
10      m = c;
11    }
12
13    return m;
14  }
15
16  int main() {
17    char c;
18
19    printf("Entrez un caractère : ");
20    c = getchar();
21    printf("%c\n", maj(c));
22
23    return 0;
24  }
```

Listing 6 – Programme `maj`

12. Recopiez, compilez et exécutez le programme du listing 6 ;
13. Que fait ce programme ? Expliquez notamment la ligne 5 et les lignes 7 à 11 ;
14. Écrivez une fonction **int** est_min(**char** c) qui renvoie 0 (`vrai`) si c est une lettre minuscule et 1 (`faux`) sinon ;
15. Modifiez le programme `maj` pour qu'il fasse appel à est_min ;
16. Modifiez le programme pour qu'il convertisse une chaîne de caractères en majuscules.

Partie 3 : Make

`Make` est un outil permettant d'automatiser la compilation de programme. Il est utile notamment lorsqu'un projet contient un grand nombre de fichiers à compiler. Seuls les fichiers devant être recompilés le sont, les fichiers non modifiés ne sont pas recompilés.

Les règles de compilation sont écrites dans un fichier nommé `makefile` ou `Makefile`.

La structure d'une règle dans un makefile est la suivante :

```
cible : liste_des_dependances
<TAB>commande1
<TAB>commande2
```

Le nom de la cible correspond au nom du fichier produit par les commandes `commande1` et `commande2` exécutées en séquence.

`liste_des_dependances` correspondant aux fichiers utilisés pour générer la cible. Si un des fichiers de `liste_des_dependances` est plus récent que le fichier `cible`, alors les commandes `commande1` et `commande2` sont exécutées. Sinon, aucune commande n'est exécutée.

Remarque. *De manière générale :*

- *`liste_des_dependances` peut faire référence à des fichiers ou bien à d'autres règles make ;*
- *`cible` ne correspond pas forcément à un fichier. Dans ce cas, les commandes associées sont systématiquement exécutés.*

```
1  all : bonjour
2
3  bonjour: bonjour.c
```

```
4       gcc bonjour.c -o bonjour
5
6   clean :
7       rm -f bonjour
```

Listing 7 – Makefile pour le programme `bonjour`

Dans le listing 7, on peut voir le `makefile` permettant de recompiler le programme `bonjour` si le fichier `bonjour.c` a été modifié.

1. Ouvrez un éditeur de texte et recopiez-y le listing 7. Sauvegardez le tout dans un fichier nommé `makefile` dans le répertoire `tpc` ;
2. Exécutez la commande **`make`** :

   ```
   $ make
   gcc bonjour.c -o bonjour
   $ ls
   bonjour* bonjour.c Makefile
   ```

   On peut voir que make a lancé automatiquement le compilateur pour produire l'exécutable `bonjour.c` ;
3. Ré-exécutez la commande :

   ```
   $ make
   make: Nothing to be done for 'all'.
   ```

   L'exécutable `bonjour` étant à jour vis-à-vis de `bonjour.c`, `make` n'a pas relancé sa compilation ;
4. Exécutez la commande **`make clean`** :

   ```
   $ make clean
   rm -r bonjour
   $ ls
   bonjour.c Makefile
   ```

On y voit apparaître, 3 règles :

— la première règle apparaissant dans le `makefile` est la règle interprétée par défaut ; son nom peut être choisi librement, mais elle est généralement appelée `all` ;

— la seconde règle est une règle spécifique décrivant comment construire le fichier `bonjour` à partir du fichier `bonjour.c` ;

— la troisième règle permet de nettoyer (*`clean`*) les fichiers générés par les autres règles du `makefile` ; par convention elle s'appelle `clean`.

5. Exécutez la commande **`make clean`** suivie de la commande **`ls`** ; Que constatez vous ?
6. Rajouter une règle au `makefile` pour compiler le programme `prog` ;
7. Doit-on modifier la règle `all` ? Doit-on modifier la règle clean ?
8. Comment faire pour dire à make de ne compiler que le programme `bonjour` ou que le programme `prog`

Partie 4 : Annexes

Dec	Hex	Oct	Car	Signification
0	0x00	000	NUL	*Null* (nul, inexistant)
1	0x01	001	SOH	*Start of Header* (début d'en-tête)
2	0x02	002	STX	*Start of Text* (début du texte)
3	0x03	003	ETX	*End of Text* (fin du texte)
4	0x04	004	EOT	*End of Transmission* (fin de transmission)
5	0x05	005	ENQ	*Enquiry (End of Line)* (demande, fin de ligne)
6	0x06	006	ACK	*Acknowledge* (accusé de réception)
7	0x07	007	BEL	*Bell* (caractère d'appel)
8	0x08	010	BS	*Backspace* (espacement arrière)
9	0x09	011	TAB	*Horizontal Tab* (tabulation horizontale)
10	0x0A	012	LF	*Line Feed* (saut de ligne)
11	0x0B	013	VT	*Vertical Tab* (tabulation verticale)
12	0x0C	014	FF	*Form Feed* (saut de page)
13	0x0D	015	CR	*Carriage Return* (retour chariot)
14	0x0E	016	SO	*Shift Out* (fin d'extension)
15	0x0F	017	SI	*Shift In* (démarrage d'extension)
16	0x10	020	DLE	*Data Link Escape*
17	0x11	021	DC1	*Device Control 1 to 4* (DC1 et DC3 sont
18	0x12	022	DC2	généralement utilisés pour coder XON et
19	0x13	023	DC3	XOFF dans un canal de communication
20	0x14	024	DC4	duplex)
21	0x15	025	NAK	*Negative Acknowledge* (accusé de réception négatif)
22	0x16	026	SYN	*Synchronous Idle*
23	0x17	027	ETB	*End of Transmission Block* (fin du bloc de transmission)
24	0x18	030	CAN	*Cancel* (annulation)
25	0x19	031	EM	*End of Medium* (fin de support)
26	0x1A	032	SUB	*Substitute* (substitution)
27	0x1B	033	ESC	*Escape* (échappement)
28	0x1C	034	FS	*File Separator* (séparateur de fichier)
29	0x1D	035	GS	*Group Separator* (séparateur de groupe)
30	0x1E	036	RS	*Record Separator* (séparateur d'enregistrement)
31	0x1F	037	US	*Unit Separator* (séparateur d'unité)
32	0x20	040	SP	*Space* (espace)
33	0x21	041	!	Point d'exclamation
34	0x22	042	"	Guillemet droit
35	0x23	043	#	Dièse ou signe numéro
36	0x24	044	$	Dollar
37	0x25	045	%	Pourcent
38	0x26	046	&	Esperluette ou perluète
39	0x27	047	’	Apostrophe ou guillemet fermant simple ou accent aigu
40	0x28	050	(	Parenthèse ouvrante
41	0x29	051	)	Parenthèse fermante
42	0x2A	052	*	Astérisque
43	0x2B	053	+	Plus
44	0x2C	054	,	Virgule
45	0x2D	055	-	Moins ou tiret ou trait d'union
46	0x2E	056	.	Point
47	0x2F	057	/	*Slash* (barre oblique)
48	0x30	060	0	
49	0x31	061	1	
50	0x32	062	2	
51	0x33	063	3	
52	0x34	064	4	
53	0x35	065	5	
54	0x36	066	6	
55	0x37	067	7	
56	0x38	070	8	
57	0x39	071	9	
58	0x3A	072	:	Deux-points
59	0x3B	073	;	Point-virgule
60	0x3C	074	<	Inférieur
61	0x3D	075	=	Égal
62	0x3E	076	>	Supérieur
63	0x3F	077	?	Point d'interrogation
64	0x40	100	@	Arobase ou A commercial
65	0x41	101	A	
66	0x42	102	B	
67	0x43	103	C	
68	0x44	104	D	
69	0x45	105	E	
70	0x46	106	F	
71	0x47	107	G	
72	0x48	110	H	
73	0x49	111	I	
74	0x4A	112	J	
75	0x4B	113	K	
76	0x4C	114	L	
77	0x4D	115	M	
78	0x4E	116	N	
79	0x4F	117	O	
80	0x50	120	P	
81	0x51	121	Q	
82	0x52	122	R	
83	0x53	123	S	
84	0x54	124	T	
85	0x55	125	U	
86	0x56	126	V	
87	0x57	127	W	
88	0x58	130	X	
89	0x59	131	Y	
90	0x5A	132	Z	
91	0x5B	133	[	Crochet ouvrant
92	0x5C	134	\	*backslash* ou *antislash* (barre oblique inversée)
93	0x5D	135	]	Crochet fermant
94	0x5E	136	^	Accent circonflexe
95	0x5F	137	_	*underscore* (tiret bas ou souligné)
96	0x60	140	‘	Accent grave
97	0x61	141	a	
98	0x62	142	b	
99	0x63	143	c	
100	0x64	144	d	
101	0x65	145	e	
102	0x66	146	f	
103	0x67	147	g	
104	0x68	150	h	
105	0x69	151	i	
106	0x6A	152	j	
107	0x6B	153	k	
108	0x6C	154	l	
109	0x6D	155	m	
110	0x6E	156	n	
111	0x6F	157	o	
112	0x70	160	p	
113	0x71	161	q	
114	0x72	162	r	
115	0x73	163	s	
116	0x74	164	t	
117	0x75	165	u	
118	0x76	166	v	
119	0x77	167	w	
120	0x78	170	x	
121	0x79	171	y	
122	0x7A	172	z	
123	0x7B	173	{	Accolade ouvrante
124	0x7C	174	|	Barre verticale
125	0x7D	175	}	Accolade fermante
126	0x7E	176	~	Tilde
127	0x7F	177	DEL	*Delete* (effacement)

FIGURE 1 – Table de caractère ASCII