PCA

TP1

Hai Nam TRAN

hai-nam.tran@univ-brest.fr

Exercice 1 : Traitement des chaîne de caractères

- **Le langage C ne dispose pas de type spécialisé pour représenter les chaînes de caractères**

```
string chaine = "Hello World";
```

  - **Des tableaux unidimensionnels de caractères simples sont utilisés à cet effets**
  - **La déclaration d'un objet de type chaîne de caractères sera donc de la forme :**

```
char chaine[80];
```

  - **La fin d'une chaîne de caractères est délimitée par un caractère nul \0**

u	n	e		c	h	a	i	n	e	\0	

  - **Une chaîne de caractères constante est une séquence de caractères encadrés par des guillemets**

Exercice 1 : Traitement des chaîne de caractères

- **Dans ce TP, nous vous proposons d'écrire des fonctions de traitement de chaîne de caractères**

```
/*Longueur d'une chaine*/
int STRLEN(char chaine[]);

/*Copie d'une chaine*/
void STRCPY(char chaineDest[], chaineSrc[]);

/*Concaténation de deux chaînes*/
void STRCAT(char chaine1[], char chaine2[]);

/*Comparaison de 2 chaînes*/
int STRCMP(char chaine1[], char chaine2[]);

/*Suppression d'une lettre dans un texte*/
int supLettre(char lettre, char chaine[]);
```

Rappel : Initialisation des tableaux de caractères

Initialisation des tableaux de caractères

Les valeurs initiales d'une chaîne de caractères peuvent être définies à l'aide d'une chaîne constante.

```
char msg[50]="un message\n";
char msg[]="le compilateur determine la taille";
```

Remarque : il faut penser à prévoir un élément pour le caractère de fin de chaîne \0. Ainsi, l'initialisation suivante est mauvaise :

```
char bad[4]="abcd";
```

Attention : les deux déclarations suivantes ne sont pas équivalentes :

```
char *m="message\n"; /* pointeur sur une chaine constante */
char m[]="message\n"; /* copie de la chaine constante et
                         allocation de memoire pour le tableau */
```

Rappel : Modifier une chaîne

```
char chaine[50] = "Hello World";
chaine = "Hello";
```

C11/18 Standard - 6.5.16 Assignment operators

An assignment operator shall have a modifiable lvalue as its left operand

C11/18 Standard - 6.3.2.1 Modifiable lvalue

A modifiable lvalue is an lvalue that does not have array type

```
char chaine[50] = "Hello World";
strcpy(chaine, "Hello");
```

Rappel : Comparer 2 chaînes de caractères

- **strcmp**
  - **0 : les deux chaines sont identiques (mêmes lettres, même longueur)**
  - **1 : chaine1 est "lexicographiquement" supérieure à chaine2**
  - **-1 : chaine1 est "lexicographiquement" inférieure à chaine2**

```
strcmp("a", "a"); // returns 0 as ASCII value of "a" and "a" are same i.e 97

strcmp("a", "b"); // returns -1 as ASCII value of "a" (97) is less than "b" (98)

strcmp("a", "c"); // returns -1 as ASCII value of "a" (97) is less than "c" (99)

strcmp("z", "d"); // returns 1 as ASCII value of "z" (122) is greater than "d" (100)

strcmp("abc", "abe"); // returns -1 as ASCII value of "c" (99) is less than "e" (101)

strcmp("apples", "apple"); // returns 1 as ASCII value of "s" (115) is greater than "\0" (101)
```

https://overiq.com/c-programming-101/the-strcmp-function-in-c/

Rappel : http://www.asciitable.com/

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

- **Vérifier la présence d'un caractère qui n'est pas une lettre ? → valeurs ASCII**

Exercice 1 : Traitement des chaîne de caractères

- **Fichier**
  - chaine.c
  - chaine.h

```
int STRLEN(char chaine[]);
void STRCPY(char chaineDest[], chaineSrc[]);
void STRCAT(char chaine1[], char chaine2[]);
int STRCMP(char chaine1[], char chaine2[]);
int supLettre(char lettre, char chaine[]);
```

  - main.c

```
...
#include "chaine.h"

int main() {
        /* Tests */
}
```

- **Compilation**

```
$: gcc chaine.c main.c -o main
$: ./main
```

Rappel : Test

- **Tests**
  - **Automatisé**

> *"Faire sortir l'étudiant par la fenêtre s'il utilise scanf pour faire les tests"*
>
> *Le prof, 2020*

  - **Complète**
    - Cas généraux
    - Cas spéciaux
  - **Descriptive**
    - Comprendre ce qui est testé, la méthode de test, et les résultats
- **Au début, je vous demande**
  - **Un test/fonction**
  - **Terminer le test d'une fonction avant d'implémenter une autre**

Rappel : Test

- **Un test doit afficher les informations suivantes**
  - [Test number] - [Description]
  - Input
  - Expected Output
  - Ouput

```
Test 01 : Function STRLEN
Input: Hello World
Expected Output: Longueur = 11
Output: Longueur = 11
```

Exercice 2 : Vérification de la "force" d'un mot de passe

- **Écrire une fonction qui vérifie si un mot de passe a les caractéristiques suivantes :**
  - **Longueur supérieure à 7 caractères**
  - **Présence d'un caractère qui n'est pas un lettre**
  - **La fonction retourne 1 si le mot de passe est correct, 0 sinon**

```
int verifyPwd(char pwd[]);
```

Exercice 2 : Vérification de la "force" d'un mot de passe

- **Fichier**
  - chaine.c
  - chaine.h

```
int STRLEN(char chaine[]);
void STRCPY(char chaineDest[], chaineSrc[]);
void STRCAT(char chaine1[], char chaine2[]);
int STRCMP(char chaine1[], char chaine2[]);
int supLettre(char lettre, char chaine[]);
int verifyPwd(char pwd[]);
```

  - main.c

```
...
#include "chaine.h"

int main() {
        /* Tests */
}
```

- **Compilation**

```
$: gcc chaine.c main.c -o main
$: ./main
```

Exercice 3 : Compilation séparée et utilitaire make

- **Fichier**
  - chaine.c
  - chaine.h

```
int STRLEN(char chaine[]);
void STRCPY(char chaineDest[], chaineSrc[]);
void STRCAT(char chaine1[], char chaine2[]);
int STRCMP(char chaine1[], char chaine2[]);
int supLettre(char lettre, char chaine[]);
int verifyPwd(char pwd[]);
```

  - main.c

```
...
#include "chaine.h"

int int main() {
        /* Tests */
}
```

  - Makefile

```
program : ...
...
```

- **Compilation**

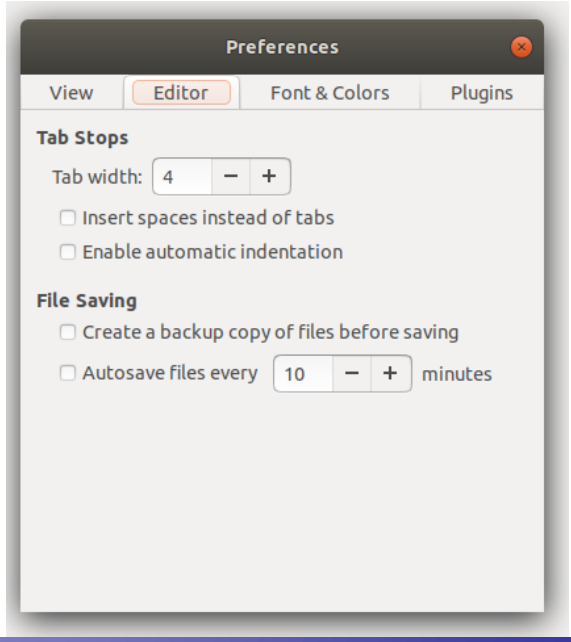
```
$: make
```

Exercice 3 : Compilation séparée et utilitaire make

- **Un Makefile est composé d'un ensemble de règles de la forme :**

```
target [target ...]: [component ...]
    [command]
    ...
    [command]
```

- **Il est important de se rendre compte que l'espacement derrière les command doit impérativement commencer par une tabulation**
  - Configurer bien votre editor de texte !!!



Rappel : Makefile

- **Fichier de règles génériques (TD1)**
  - **Il est possible d'écrire des fichiers de règles "génériques" pour make. Ces fichiers sont adaptables facilement à plusieurs projets de développement logiciel. En voici ci-dessous un exemple simple.**

```
src = main.c chaine.c
obj = $(src:.c=.o)

prog : $(obj)
	gcc -o prog $(src)

%.o : %.c
	gcc -c $< -o $@

clean :
	rm -f prog
	rm -f *.o
```

  - **Un petit tutoriel : https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/**

Makefile v1

```
tp3: main.c func.c
	gcc -o tp3 main.c func.c -Wall -lm
```

- Just need to type "**make**" or "**make tp3**"
- **Problem** : any changes in either main.c or func.c require recompiling all the files in the list of dependencies

Makefile v2

```
CC=gcc
CFLAGS=-Wall
LDLIBS=-lm

tp3: main.o func.o
        $(CC) -o tp3 main.o func.o $(CFLAGS) $(LDLIBS)
```

- **Macro CC , CFLAGS, LDLIBS**
  - More information : http://www.gnu.org/software/make/manual/html_node/Implicit-Variables.html
- **main.o and func.o in the dependency list**
  - → **we first compile the .c versions individually (automatically)**
  - → **avoid recompiling if there is no change**
- **Normally, this version of makefile is good enough for a simple project**
- **Problem : if only func.h changes → no recompile !**

Makefile v2.1

```
CC=gcc
CFLAGS=-Wall
LDLIBS=-lm

func.o : func.c func.h
	$(CC) -c -o func.o func.c $(CFLAGS)

main.o : main.c func.h
	$(CC) -c -o main.o main.c $(CFLAGS)

tp3: main.o func.o
	$(CC) -o tp3 main.o func.o $(CFLAGS) $(LDLIBS)
```

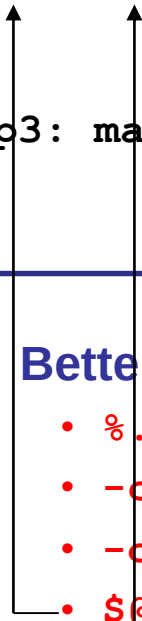
- Solution : say that other files depend on func.h
- Better but a little bit long ...

Makefile v3

```
CC=gcc
CFLAGS=-Wall
LDLIBS=-lm
DEPS=func.h

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

tp3: main.o func.o
	$(CC) -o tp3 main.o func.o $(CFLAGS) $(LDLIBS)
```



- Better... but look like magic...
  - `%.o: %.c $(DEPS)` : a rule applies to all .o files
  - `-c` : generate the object file
  - `-o` : output
  - `$@` : file named on the left side of the ":"
  - `$<` : first item in the dependency list (.c file in this case)

Makefile v4

```
CC=gcc
CFLAGS=-Wall
LDLIBS=-lm
DEPS=func.h
OBJ=main.o func.o

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

tp3: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS) $(LDLIBS)

clean:
	rm -f *.o
```

- **Look quite good now for a small or medium-sized project**
- **Still can be improved (to explore)**
  - Organize .h files in a folder : .../include
  - Put .o files in a folder : .../obj
  - .PHONY target and its usage