

# LIFAP1 : ALGORITHMIQUE ET PROGRAMMATION IMPÉRATIVE, INITIATION

1

COURS 7 : Les chaînes de caractères

## OBJECTIFS DE LA SÉANCE

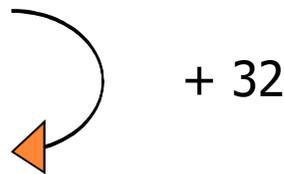
- Apprendre à manipuler les chaînes de caractères
- Voir un exemple de conception utilisant largement les fonctionnalités des chaînes de caractères
- Que se passe-t-il quand je clique sur compiler ou exécuter mon programme ?

# PLAN

- Les chaînes de caractères
  - Déclaration
  - Accès
  - Manipulation
  - En algorithmique et en C
- Exemple de conception : Le jeu du pendu
- La chaîne de traitement de l'écriture à l'exécution du code en passant par le débogage

# PARLONS D'ABORD DES CARACTÈRES

- Caractère : type simple → peut être renvoyé par une fonction C (instruction return)
- “char” représente **une** lettre ou **un** caractère spécial (mis entre quotes ‘ ’ , touche 4)
  - 'a' ou '\n' (saut de ligne)
- Le “char” est vraiment codé comme un numéro entre 0 et 255. C'est le codage ASCII
- 'A' == 65, 'Z' == 90
- 'a' == 97, 'z' == 122



# LE CODE ASCII

- Chaque caractère a un « code » unique

- Entier entre 0 et 255

- Exemple

- E 69
- x 120
- e 101
- m 109
- p 112
- l 108
- e 101

32		64	@	96	`	48	0	80	P	112	p
33	!	65	A	97	a	49	1	81	Q	113	q
34	"	66	B	98	b	50	2	82	R	114	r
35	#	67	C	99	c	51	3	83	S	115	s
36	\$	68	D	100	d	52	4	84	T	116	t
37	%	69	E	101	e	53	5	85	U	117	u
38	&	70	F	102	f	54	6	86	V	118	v
39	'	71	G	103	g	55	7	87	W	119	w
40	(	72	H	104	h	56	8	88	X	120	x
41	)	73	I	105	i	57	9	89	Y	121	y
42	*	74	J	106	j	58	:	90	Z	122	z
43	+	75	K	107	k	59	;	91	[	123	{
44	,	76	L	108	l	60	<	92	\	124	
45	-	77	M	109	m	61	=	93	]	125	}
46	.	78	N	110	n	62	>	94	^	126	~
47	/	79	O	111	o	63	?	95	_	127	

# PARLONS D'ABORD DE CARACTÈRES

- On peut comparer des caractères

```
bool estMajuscule(char c)
{
    if ((c >= 'A') && (c <= 'Z'))
        return true;
    else return false;
}
```

- On peut les soustraire

```
int char2decimal(char c)
{
    if (c >= '0' && c <= '9')
        return c - '0';
    else return -1;
}
```

# LA CHAÎNE DE CARACTÈRES

- Pour représenter des phrases ou des ensembles de caractères
  - Algo : chaîne [10] de caractères
  - C/C++ : char mot[10]
- On a besoin de connaître la taille
  - On peut mettre des chaînes plus petites que la taille maximale -1 dans le tableau
    - On marque la fin de la chaîne avec le caractère `'\0'`
    - Pour les caractères numériques : pas la valeur mais le caractère !!
      - Pas la valeur 0 mais vraiment caractère '0' !!!

# CARACTÈRE / CHAÎNE DE CARACTÈRES

- Notez que "a" est la chaîne de caractères et 'a' est le caractère.
- Exemple 1 :
  - `char mot_a[2];` déclaration
  - `mot_a[0] = 'a';` remplissage de la 1<sup>ière</sup> case
  - `mot_a[1] = '\0';` caractère de fin de chaîne
- Exemple 2 :
  - `char let_a[1];`
  - `let_a[0] = 'a'...` est un tableau avec une seule lettre... pas une chaîne de caractères

'a' = char

"a" = mot\_a = char[2] = 

'a'	0
-----	---

let\_a = char[1] = 

'a'
-----

# CHAÎNE DE CARACTÈRES : SYNTHÈSE

- Déclaration
  - `char mot[100];`
- Lire du clavier
  - `cin >> mot;`
- Accès à la lettre de la case 4 :
  - `mot[4]`
- Toutes les cases ne sont pas nécessairement remplies
- Caractère de fin de chaîne : `'\0'`

## CHAÎNE DE CARACTÈRES DANS LES FONCTIONS / PROCÉDURES

- En C, on ne peut pas renvoyer comme résultat une chaîne de caractères → c'est un tableau.
  - **NON :**

```
char [10] saisir_phrase(void)
{ char mot[10]; .... return mot; }
```
- Elle sont toujours données/résultats quand elle sont en paramètre, donc on met pas &
  - Déclaration : 

```
int tailleMot(char mot[10]);
```
  - C'est le cas pour tous les tableaux !
  - **NON :**

```
int tailleMot(char & mot[10]);
```

## OPÉRATIONS DE BASE SUR LES CHAÎNES DE CARACTÈRES

- Il existe un certain nombre d'opérations prédéfinies sur les chaînes de caractères
- Dans la **bibliothèque** `<string.h>`  
(en anglais chaîne de caractères = string)
- Du même ordre que celles existant sur les entiers, les caractères...
- Quelques exemples...

# COMPARAISON DE CHAÎNES

- Et si on veut écrire une fonction qui trouve l'ordre (dictionnaire) entre deux mot ?
  - « aurevoir » avant « bonjour »
  - « bon » avant « bonjour »
  - ...

```
int strcmp(char a[100], char b[100])
```

Renvoie 0 si  $a == b$

Renvoie 1 si  $a > b$

Renvoie -1 si  $a < b$

# LONGUEUR DE LA CHAÎNE

- Fonction prédéfinie `strlen` (pour **string length**)
- Compter nombre de lettres
  - `strlen(mot)`
  - ou `for(i = 0; mot[i] != '\0'; i++)`
- Le grand problème est la condition de terminaison
  - Que passe-t-il s'il n'y a pas de `'\0'` dans `mot` dans l'exemple ci-dessus ?
  - C'est la cause de beaucoup de problèmes...

# CONCATÉNATION DE CHAÎNES

- Fonction prédéfinie **strcat** (**string concat**)
- Permet d'ajouter un chaîne à la suite d'une autre :  
"bon" + "jour" = "bonjour"

```
void strcat(mot1[100], mot2[100])
```

- Met le mot2 à la suite de mot1
- Rajoute automatiquement le '\0' en fin de chaîne reconstituée

## COPIE DE CHAÎNE

- On n'a pas le droit d'écrire

```
char mot[100];  
mot="hello"; /* NON */
```

- Pour copier une chaîne dans une autre il faut impérativement passer par la fonction **strcpy** (**string copy**)
  - Exemple : `strcpy(mot,"hello");`

# PLAN

- Les chaînes de caractères
  - Déclaration
  - Accès
  - Manipulation
  - En algorithmique et en C
- Exemple de conception : Le jeu du pendu
- La chaîne de traitement de l'écriture à l'exécution du code en passant par le débogage

# EXEMPLE D'UTILISATION : JEU DU PENDU

- On souhaite écrire un mini-jeu qui demande à l'utilisateur de trouver toutes les lettres d'un mot choisi par une autre personne (jeu du pendu)
- Au début de la partie, la solution comporte autant d'emplacements vides que de lettres dans le mot à trouver.
- Lorsque le joueur propose une lettre, il suffit de vérifier son existence dans le mot à trouver et de la faire apparaître dans la solution. Le jeu s'arrête lorsque le joueur a trouvé toutes les lettres ou après 4 échecs.
  - Exemple : trouver toto / solution - - - -  
saisie lettre o / solution -o-o  
saisie lettre t / solution toto  
mot trouvé donc gagné

# ANALYSER LE PROBLÈME

- Vous avez un problème, comment le résoudre ?
  - Diviser en sous-problèmes plus simples
  - Penser :
    - De quelle(s) information(s) avez vous besoin ?
      - Variables
    - De quelle manière sera manipulée cette information ?
      - Opérations, fonctions, procédures
  - Codage :  
en programmation, on représente beaucoup de choses avec des nombres...
    - Comment on va représenter l'information ou l'état... ?

## SOLUTION (1/4)

- Écrivez la procédure saisie() qui demande à une autre personne de choisir un mot à trouver.
  - Déclarer une chaîne de caractères pour stocker le mot proposé
  - Saisie du mot (cin >> mot)
  - Restitution de sa valeur (paramètre en donnée / résultat par défaut)

## SOLUTION (2/4)

- Écrivez la fonction `creer_solution()` qui remplit la chaîne de caractères représentant la solution par des tirets.
  - Chaque tiret représente une lettre du mot à trouver.
  - La fonction renvoie le nombre de lettres à trouver.
  - Idée de codage !

## SOLUTION (3/4)

- Écrivez la fonction `solution()` qui recherche la lettre passée en paramètre.  
Pour chaque occurrence, remplacez le tiret correspondant de la solution.
- Cette fonction renvoie le nombre de remplacements effectués.
  - Idée : qu'est ce que on a besoin de manipuler...
    - Chaîne initiale
    - Chaîne résultat
    - Lettre à rechercher
    - Comparaison de caractères...

## SOLUTION (4/4)

- Écrivez la fonction principale qui enchaîne les différentes étapes.
- Lorsque la fonction solution renvoie 0, l'utilisateur perd un essai, puisqu'il n'a pas trouvé de lettre.
- Le nombre de lettres restant à trouver se déduit facilement dans le cas où une lettre a été découverte.
- La solution et le nombre d'essais restants sont affichés à chaque tentative.
- Le jeu se termine lorsqu'il ne reste plus d'essais ou que l'utilisateur a trouvé le mot.

# DÉMO

- Voilà le programme correspondant !!

# CONCLUSION

- Les chaînes de caractères
  - Tableau contenant des caractères
  - Caractère de fin de ligne
  - Opérations prédéfinies
    - Dans la bibliothèque `string.h` principalement