

## 3. C# - Valider un mot de passe

---

### Valider un mot de passe

---

- [Valider un mot de passe](#)
- [Compétence](#)
- [Source](#)
- [GitHub](#)
- [Valider un mot de passe](#)
- [Objectif](#)
- [Travail à faire](#)
- [Saisie du mot de passe](#)
- [Validation de la longueur](#)
- [Récupération des caractères du mot de passe](#)
- [Validation du 1er caractère](#)
- [Validation du 4ème caractère](#)
- [Utilisation d'un tableau](#)
- [Caractère spécial](#)
- [Proposition de solution](#)

## Compétence

---

B3.3. Sécurisation des équipements et des usages des utilisateurs

## Source

---

Mathieu Frétière

## GitHub

---

<https://github.com/LiliwoL/CSharp-ValidationRobustesseMotDePasse>

---

## Valider un mot de passe

---

## Objectif

---

Le programme va valider un mot de passe saisi par l'utilisateur, c'est à dire vérifier qu'il est assez long et complexe, c'est-à-dire qu'il contient bien des majuscules, des chiffres, des caractères spéciaux, etc.

## Travail à faire

---

1. Démarrer Visual Studio
2. Configurez-le en Français si besoin : menu **Tools / Options** / General / French / OK
3. Créer une **nouvelle application console**  
Nom : **ValidationRobustesseMotDePasse**

## Saisie du mot de passe

---

- Copier le code suivant qui demande la saisie du mot de passe :

```
class Program
{
    public static void Main(string[] args)
    {

        // saisie du mot de passe
        Console.WriteLine("Saisissez un mot de passe à 4 caractères");
        String mdp = Console.ReadLine();

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

## Validation de la longueur

---

- Travail à faire : à la suite du programme, obtenez la longueur du mot de passe et affichez-là :

```
// vérification de la longueur
int longueur = mdp.Length;
Console.WriteLine("Longueur : " + longueur);
```

- Testez le programme
- Travail à faire : complétez le code afin de vérifier que la longueur du mdp est supérieure ou égale à 4 :

```
// booléen qui indique si le mdp est valide ou pas
Boolean valide = false;

// vérification si longueur supérieure ou égale à 4
```

```

if (    ??? à compléter    )
{
    // valide passe à true
    valide = true;
}
// test si le booléen est vrai ou faux
if (    ??? à compléter    )
{
    // affichage : mdp valide
}
else
{
    // affichage : mdp pas valide
}

```

- Testez plusieurs cas : un mot de passe de 4 lettres, un mot de passe de 3 lettres, puis de 5 lettres.

## Récupération des caractères du mot de passe

---

- Travail à faire : compléter le code afin d'afficher successivement les 4 caractères du mdp.

```

// à la suite, après le test de la longueur
// récupération des lettres du mot de passe
// carac1 est le 1er caractère du mdp
char carac1 = mdp[0];

// carac2 est le 2ème
char carac2 = mdp[1];

// etc.
char carac3 = mdp[2];

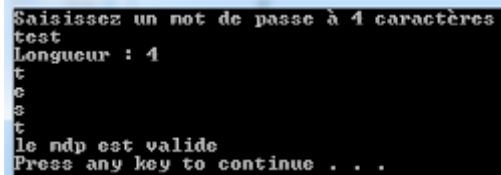
```

```
char carac4 = mdp[3];

// à compléter : afficher carac1, carac2, etc.

// à la fin : mettez le code qui teste le booléen valide
```

Résultat attendu:



```
Saisissez un mot de passe à 4 caractères
test
Longueur : 4
t
e
s
t
le mdp est valide
Press any key to continue . . .
```

## Validation du 1er caractère

---

- Objectif : on souhaite que le 1er caractère soit une majuscule et le programme doit le vérifier.  
-Travail à faire : écrivez le code qui récupère le code ASCII du 1er caractère et qui l'affiche:

```
// vérification du 1er caractère
int codeASCII1 = (int)carac1;
Console.WriteLine("le 1er code est " + codeASCII1);
```

- Testez

- Ensuite, testez si ce code codeASCII1 est bien compris entre 65 ('A') et 90 ('Z').

65 ('A') et 90 ('Z').

);

Dec	Hx	Oct	Html	Chr
64	40	100	&#64;	@
65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D
69	45	105	&#69;	E
70	46	106	&#70;	F
71	47	107	&#71;	G
72	48	110	&#72;	H
73	49	111	&#73;	I
74	4A	112	&#74;	J
75	4B	113	&#75;	K
76	4C	114	&#76;	L
77	4D	115	&#77;	M
78	4E	116	&#78;	N
79	4F	117	&#79;	O
80	50	120	&#80;	P
81	51	121	&#81;	Q
82	52	122	&#82;	R
83	53	123	&#83;	S
84	54	124	&#84;	T
85	55	125	&#85;	U
86	56	126	&#86;	V
87	57	127	&#87;	W
88	58	130	&#88;	X
89	59	131	&#89;	Y
90	5A	132	&#90;	Z
91	5B	133	&#91;	[
92	5C	134	&#92;	\
93	5D	135	&#93;	]
94	5E	136	&#94;	^
95	5F	137	&#95;	_

```
// vérification du 1er caractère
int codeASCII1 = (int)carac1;
Console.WriteLine("le 1er code est " + codeASCII1);

Boolean valideCar1 = false;

// si le code ASCII est compris entre 65 et 90
```

```
if ( ???      ???      ???  à compléter)
{
    valideCar1 = true;
}

// à la fin du programme :
// test si les booléens sont vrais
if (valide == true  && valideCar1 == true)
{
    // affichage : mdp valide
}
// sinon : mdp pas valide
```

- Testez plusieurs mots de passe, valides ou invalides.

## Validation du 4ème caractère

---

- Objectif : le 4ème caractère doit être un chiffre et le programme doit le vérifier.

- Travail à faire : modifier le programme afin qu'il teste que le 4ème caractère soit bien un chiffre. Utilisez la table ASCII ci-dessous.

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

- Testez plusieurs mots de passe, valides ou invalides.

## Utilisation d'un tableau

Objectif : afficher toutes les lettre d'un mot de passe plus long que 4 caractères.

Explication : en fait un String est un tableau de caractères.

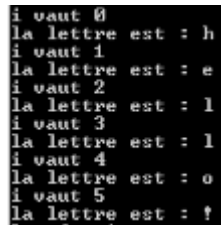
On peut donc récupérer toutes les lettres comme les éléments d'un tableau.



Travail à faire : modifier la programme afin d'afficher le mot de passe avec une boucle for :

```
for (int i = 0 ; i < mdp.Length ; i++)  
{  
    Console.WriteLine("i vaut " + i);  
    Console.WriteLine("la lettre est : "+ mdp[i]);  
}
```

Testez avec un mot de passe de longueur quelconque. Par exemple « hello! »



```
i vaut 0  
la lettre est : h  
i vaut 1  
la lettre est : e  
i vaut 2  
la lettre est : l  
i vaut 3  
la lettre est : l  
i vaut 4  
la lettre est : o  
i vaut 5  
la lettre est : !
```

## Caractère spécial

---

Objectif : le mot de passe (quel que soit sa longueur) doit contenir au moins un caractère spécial, quelle que soit sa position.

Le caractère spécial peut-être : ! " # \$ % & ' ( ) etc.

Travail à faire : modifier la programme pour qu'il vérifie que le mot de passe contienne au moins un caratère spécial. Utilisez la table ASCII ci-dessous.

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

Testez plusieurs mots de passe, valides ou invalides.

## Proposition de solution

```
using System;

namespace ValidationRobusteseMotDePasse
{
```

```
class Program
{
    public static void Main(string[] args)
    {

        // saisie du mot de passe
        Console.WriteLine("Saisissez un mot de passe à 4 caractères");
        String mdp = Console.ReadLine();

        // booléen qui indique si le mdp est valide ou pas
        Boolean valide = false;

        // vérification de la longueur
        int longueur = mdp.Length;
        Console.WriteLine("Longueur : " + longueur);

        if (longueur >= 4)
        {
            valide = true;
        }
        else
        {
            Console.WriteLine("Longueur : " + longueur);
            Console.WriteLine("Longueur trop courte");

            // Console app
            System.Environment.Exit(1);
        }

        // récupération des lettres du mot de passe
    }
}
```

```
// carac1 est le 1er caractère du mdp
char carac1 = mdp[0];
// carac2 est le 2ème
char carac2 = mdp[1];
// etc.
char carac3 = mdp[2];
char carac4 = mdp[3];

// affichage des caractères pour vérifier
Console.WriteLine(carac1);
Console.WriteLine(carac2);
Console.WriteLine(carac3);
Console.WriteLine(carac4);

// vérification du 1er caractère
int codeASCI1 = (int)carac1;
Console.WriteLine("le 1er code est " + codeASCI1);

Boolean valideCar1 = false;

// si le code ASCII est compris entre 65 et 90
if (codeASCI1 >= 65 && codeASCI1 <= 90)
{
    valideCar1 = true;
}

int codeASCI4 = (int)carac4;
```

```
Boolean valideCar4 = false;

if (codeASCII4 >= 48 && codeASCII4 <= 57)
{
    valideCar4 = true;
}

// caractère spécial 33-47 91-96 123-126

// test si le booléen est vrai ou faux
if (valide == true && valideCar1 == true && valideCar4 == true)
{
    // affichage : mdp valide
    Console.WriteLine("le mdp est valide");
}
else
{
    // affichage : mdp pas valide
    Console.WriteLine("le mdp n'est pas valide");
}

Console.Write("Press any key to continue . . . ");
Console.ReadKey(true);
}
}
}
```