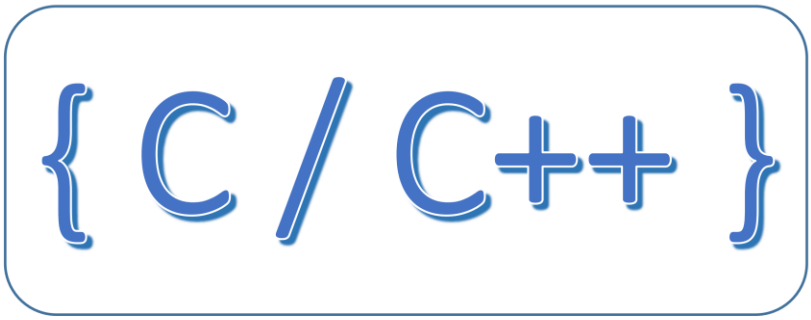
LYCÉE ST-JOSEPH HASPARREN BTS SN-IR	C++ CONSTANTES ET ENCODAGES	COURS

Tâches professionnelles :	T7.2	Réaliser une maquette, un prototype logiciel/matériel
Compétences du référentiel	C4.4	Développer un module logiciel.
Savoirs - Savoir faires	S4.7	Langages de programmation C++

Table des matières



1 EXPRESSIONS CONSTANTES TYPEES

Parfois, il est plus pratique de donner un nom à une valeur constante, plutôt que de réécrire chaque fois la constante :

```
1 const double pi = 3.1415926;
2 const char tab = '\t';
```

On peut alors utiliser ces noms au lieu des littéraux qui les ont définis :

```
1  #include <iostream>
2  using namespace std;
3
4  const double pi = 3.14159;
5  const char newline = '\n';
6
7  int main ()
8  {
9    double r=5.0;                // radius
10   double circle;
11
12   circle = 2 * pi * r;
13   cout << circle;
14   cout << newline;
15 }
```

```
31.4159
```

Pour une variable classique on peut modifier la valeur plusieurs fois dans le programme.

Pour une constante typée l'initialisation se fait une fois pour toute à la déclaration, et **sa valeur ne peut plus être modifiée ensuite**. On pourrait parler de « variable constante » !

EXERCICE : un radar mesure la distance à partir du temps mis par l'onde sonore à faire l'aller/retour. Pour tester, le temps sera saisi par l'opérateur, et la vitesse du son est une constante typée qui vaut 340,29 m/s.

2 DEFINITIONS DU PREPROCESSEUR (#DEFINE)

Un autre mécanisme pour nommer des valeurs constantes est l'utilisation de définitions de préprocesseur (héritage du C, il ne faut plus faire comme ça).

Elles ont la forme suivante: **#define** ***identifiant*** ***remplacement***

Après cette directive, toute occurrence de *identifiant* dans le code est remplacée par *remplacement* , où *identifiant* et *remplacement* sont une séquence de caractères.

Ce remplacement est effectué par le préprocesseur, et se produit avant que le programme soit compilé, provoquant ainsi un remplacement aveugle : la validité des types ou la syntaxe impliquée ne sont pas vérifiés (**à évite**r, mais historiquement très utilisé).

Notez que les lignes #define sont des directives préprocesseur, et en tant que telles sont des instructions sur une seule ligne (contrairement aux instructions C ++), et ne nécessitent pas de points-virgules (;) à la fin.

La directive s'étend automatiquement jusqu'à la fin de la ligne. Si un point-virgule est inclus dans la ligne, il fait partie de la séquence de remplacement et est également inclus dans toutes les occurrences remplacées.

```
1  #include <iostream>
2  using namespace std;
3
4  #define PI 3.14159
5  #define NEWLINE '\n'
6
7  int main ()
8  {
9    double r=5.0;
10   double circle;
11
12   circle = 2 * PI * r;
13   cout << circle;
14   cout << NEWLINE;
15
16 }
```

3 CONSTANTES LITTERALES

Les littérales sont les plus évidentes des constantes, par exemple, lorsque nous avons écrit :

```
a = 5;
```

Dans ce morceau de code le ***5*** est une ***constante littérale*** (ici littérale ne veut pas dire lettre).

Les constantes littérales peuvent être un entier, à virgule flottante, des caractères, des chaînes, un booléen, etc.

3.1 Constantes numériques entières

```
1776
707
-273
```

Ce sont des constantes numériques qui ont des valeurs entières. Remarquez qu'ils ne sont pas entre guillemets ou tout autre caractère spécial ; ils sont une simple succession de chiffres représentant un nombre entier en base décimale.
En plus des nombres décimaux (ceux que la plupart d'entre nous utilisent tous les jours), C ++ permet d'utiliser des nombres en octal (base 8) et des nombres hexadécimaux (base 16) en tant que constantes littérales :

- **Pour l'octal**, les chiffres sont précédés du caractère **0** (zéro)
- **Pour l'hexadécimal**, ils sont précédés par les caractères **0x** (zéro, x)
- **Pour le décimal**, le nombre est écrit tel quel.

Par exemple, les constantes littérales suivantes sont toutes équivalentes les unes aux autres :

```
75          // decimal
0113        // octal
0x4b        // hexadecimal
```

Ces constantes littérales ont un type, tout comme les variables. Par défaut, les littérales entiers sont de type ***int***.
Cependant, certains suffixes peuvent être ajoutés à un entier littéral pour spécifier un type d'entier différent :

Suffix	Type modifier
u *or* U	unsigned
l *or* L	long
ll *or* LL	long long

Unsigned peut être combiné avec l'un des deux autres dans un ordre quelconque pour former unsigned long ou unsigned long long.

Par exemple :

```
75          // int
75u         // unsigned int
75l         // long
75ul        // unsigned long
75lu        // unsigned long
```

Dans tous les cas ci-dessus, le suffixe peut être spécifié en utilisant des lettres majuscules ou minuscules (ce n'est ***pas sensible à la casse***).

3.2 Constantes numériques à virgule flottante

Ils expriment des valeurs réelles, avec des décimales et / ou exposants. Ils peuvent inclure soit un point décimal, un caractère ***e*** (qui signifie *"fois 10 puissance X"*, où *X* est une valeur entière qui suit le caractère ***e***), ou à la fois un point décimal et un caractère ***e***:

```
3.14159     // 3.14159
6.02e23     // 6.02 x 10^23
1.6e-19     // 1.6 x 10^-19
3.0         // 3.0
```

Ce sont quatre représentations de nombre avec décimales valides en C ++. Le premier numéro est PI, le second est le nombre d'Avogadro, le troisième est la charge électrique d'un électron, et le dernier est le nombre *trois* exprimé en littérale numérique à virgule flottante.

Le type par défaut pour un littéral à virgule flottante est le ***double***.

Des littéraux de type float ou long double peuvent être spécifiées en ajoutant un des suffixes suivants :

Suffix	Type
`f` *or* `F`	`float`
`l` *or* `L`	`long double`

Par exemple :

```
3.14159L    // long double
6.02e23f    // float
```

Chacune des lettres qui peuvent faire partie d'une constante numérique à virgule flottante (e, f, l) peuvent être écrites en utilisant soit des lettres majuscules ou minuscules, indifféremment.

3.3 Constantes de type caractère et chaîne

Les constantes littérales de type ***caractères*** sont entre guillemets simples (') :

Les constantes littérales de type ***chaîne de caractères*** sont entre guillemets (") :

```
'z'
'p'
"Hello world"
"How do you do?"
```

Les guillemets simples (single quotes) et les guillemets doubles (double quotes) permettent de les distinguer de possibles identificateurs de variables ou des mots clés réservés.

Notez la différence entre ces deux expressions: **x** et **'x'**
Ici, **x** seul ferait référence à un identifiant, comme le nom d'une variable, alors que **'x'** (entre quottes simples) serait référer au caractère littéral 'x' (le caractère qui représente un *x* minuscule lettre).

3.3.1 Séquences d'échappement

Dans les littérales de type caractères ou chaîne on peut également trouver des **caractères spéciaux** comme **newline (\n)** ou **tabulation (\t)**.

On les appelle aussi **séquences d'échappement**.

Ces caractères spéciaux sont tous précédés d'une barre oblique inverse ou **back slash \.**

Ci-contre une liste des codes d'échappement de caractère unique:

Plus de détails ici : http://fr.cppreference.com/w/cpp/language/escape

Escape code	Description
`\n`	newline
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\b`	backspace
`\f`	form feed (page feed)
`\a`	alert (beep)
`\'`	single quote (')
`\"`	double quote (")
`\?`	question mark (?)
`\\`	backslash (\)

Exemple : Vous devez écrire et tester un programme qui donne la sortie ci-dessous, en utilisant les affectations (en style C par exemple) de constantes littérales vues précédemment.

```
unInt vaut: 45
unDouble vaut: 45.3
unFloat vaut: 4.5e+09
unCaractere vaut: E
unAutreCaractere vaut: 6
uneChaine vaut: Bonjour                    tiens on a saute 2 tab
uneAutreChaine vaut: Re-Bonjour, tiens un \, un ', et un "
```

A titre d'exemple les premières lignes sont comme ceci :

```
// Constantes littérales

#include <iostream>
#include <string>

int main()
{
    int unInt = 45;
    std::cout << "unInt vaut: " << unInt << '\n';









}
```

3.3.2 Code ASCII

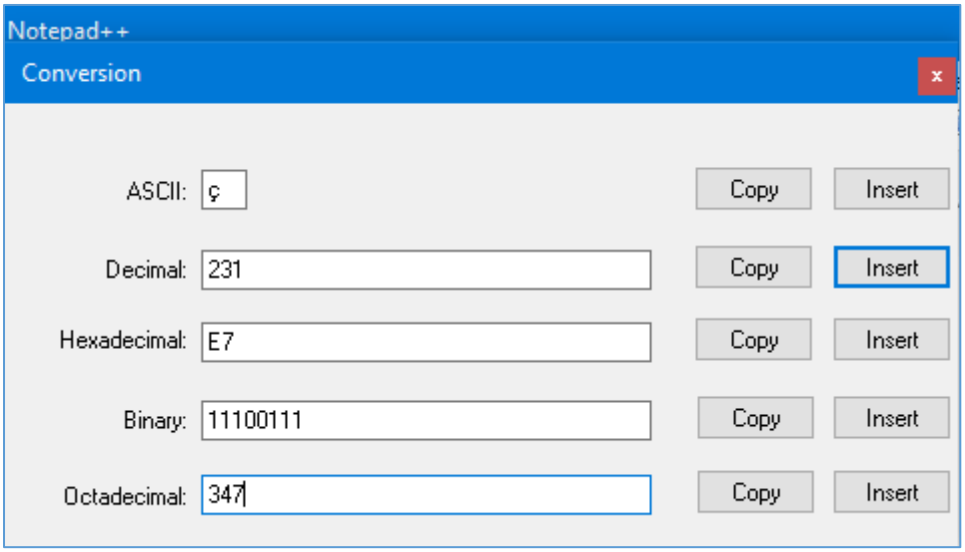
En interne, dans les ordinateurs, les caractères sont codés sous forme de codes numériques : UTF-8, Windows-1252, etc. Le plus souvent, ils utilisent une extension du système de codage de caractères de base appelé ASCII (pour plus d'info voir à https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange).

- Le code hexa 0x09 provoque une tabulation horizontale, Horizontal Tab
- Le code hexa 0x41 provoque l'affichage du caractère 'A'
- Le code 0x33 provoque l'affichage du caractère '3'
- Le code 0xE7 provoque l'affichage du caractère 'Ç'

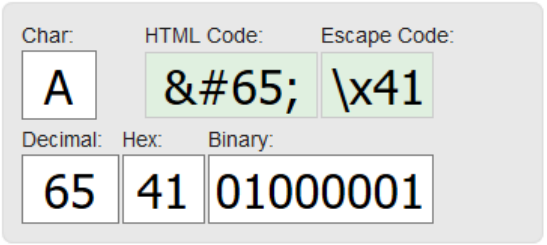
Windows-1252 (CP1252)

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9x		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Le convertisseur de Notepad++ (Compléments – Converter – Conversion Panel) permet de visualiser les codes :



Vous pouvez aussi aller voir :
http://www.rapidtables.com/code/text/ascii-table.htm



Le caractère ASCII '**A**' est codé :

- **$(01000001)_2$** , en binaire,
- ce qui peut aussi s'écrire **$(41)_{16}$** , en hexa,
- ou **$(65)_{10}$** , en décimal.

➢ On a vu que si on envoie **'\n'** sur **std ::cout** cela provoque un saut de ligne (Line Feed) ; si on envoie le code ASCII équivalent cela aura le même effet, **mais il faut convertir le code en hexadecimal**, soit **'\x09'**.

De même au lieu d'envoyer **'A'** on peut envoyer son code ASCII, soit **'\x41'**, en hexadécimal, car $(41)_{16} = (65)_{10}$

3.3.3 Affichages de caractères spéciaux dans une chaine

Nous avons vu qu'il y avait des caractères qui ont une signification spéciale (le ****, le **'** et le **"**) et qui ne sont donc pas affichés.

Si on veut afficher un ****, ou un **'**, ou un **"** on les fait précéder d'un **\ :** **** affiche : **** **\'** affiche : **'** et **\"** affiche : **"**

Mais si on voulait afficher d'autres combinaisons de ces caractères spéciaux cela devient compliqué, et on a une autre solution : on va afficher n'importe quelle série de caractères comprises entre des délimiteurs :

R"sequence(de tout, des \\\ des ''' des """ etc.**)sequence"** .

Exemples : `std::string autreChaine = R""(de tout, des \\\ des ''' des """ etc.)"";`

Ou `std::string autreChaine = R"*(de tout, des \\\ des ''' des """ etc.)*";`

Ou `std::string autreChaine = R"*%(de tout, des \\\ des ''' des """ etc.)*%";`

Plus de détails ici : http://en.cppreference.com/w/cpp/language/string_literal

EXERCICE : saisissez un programme qui, à partir d'une chaine constante, affiche :

```
\\"double backslash"\\
```
Par la méthode des séquences d'échappement

```
©¥'\"|"/'¥©
```
Par la méthode des séquences R

```
©¥'\"|"/'¥©
```
A nouveau par la méthode des codes hexa (escape code)

3.4 Autres littérales

Trois mots clés du C ++ sont de type littéral : ***true***, ***false*** et ***nullptr***

```
1 bool foo = true;
2 bool bar = false;
3 int* p = nullptr;
```

- ***true*** et ***false*** sont les deux valeurs possibles pour les variables de type bool .
- ***nullptr*** est la valeur *de pointeur NULL* (sera vu plus tard).

3.5 EXERCICE :

Donnez le programme qui va permettre d'afficher ces messages en espagnol, de lire les saisies de l'utilisateur et d'afficher le résultat ceci :

```
Console de débogage Microsoft Visual Studio
Da un número de 0 à 10: 6
¿ Cómo te llamas ? Pedro
Te llamas Pedro y elegiste el número 6
```