UNIVERSIDAD NACIONAL DE INGENIERÍA

Facultad de Ciencias

Escuela Profesional de Ciencia de la Computación

- **Curso: Fundamentos de Programación**
- **Docente:** Américo Chulluncuy Reynoso

2025-3

Sesión 6:

Cadena de caracteres I

Contenido

1. Representación en memoria: Caracteres ASCII
2. Aritmética limitada de Caracteres
3. Arreglo y cadena de caracteres
4. Puntero y cadena de caracteres
5. Funciones de entrada de datos
6. Funciones para el manejo de caracteres I

Resumen: Memoria, punteros y arreglos

- **Punteros y memoria**
  1. Si `int v=10;` es una variable, entonces &v es su **referencia** (dirección).
  2. `int *p;` declara a p como un **puntero a int**.
  3. Si hacemos p = &v, entonces *p es v. (desreferencia)

- **Punteros y arreglos**
  1. Si `int a[N];` entonces a es un puntero **const** a su primer elemento
  2. Si hacemos p = a, entonces p+i vale &a[i] y *(p+i) es a[i].
  3. Los punteros a elementos de un arreglo se pueden incrementar p++, decrementar p-- y comparar para moverse en un arreglo.

```
    char c[] = "Hola";
    c[0] = 'S';
    //c++;  ERROR el nombre de un arreglo es un puntero CONSTANTE
    cout << c <<endl;
    cout << "sizeof(c) = " << sizeof(c) <<endl; //tamaño del arreglo en bytes


    const char *d = "Mundo";
    d++; //aritmetica de punteros
    cout << d <<endl;
    cout << "sizeof(d) = " << sizeof(d) <<endl; //tamaño del puntero

    cout << *d <<endl; //desrreferencia

    cout << (void *)&d[0] <<endl; //Conversión estilo C direccion del carácter

//La conversión explícita al estilo C++ seria
// reinterpret_cast<void*>(&d[0])
```

1. Caracteres ASCII.

- ASCII (American Standard Code for Information Interchange) es un estándar que asigna un valor numérico a 128 caracteres (0-127), incluyendo letras, números, signos de puntuación y caracteres de control. (1967, primera versión, ASCII estándar)

- En 1981, **IBM desarrolló una extensión** del código ASCII (“página de código 437”), donde se incorporaron 128 caracteres nuevos, como símbolos gráficos, letras acentuadas, signos de puntuación entre otros (0 - 255).

- En la actualidad, la mayoría de los sistemas informático utilizan el código ASCII para representar caracteres, símbolos, signos y textos.

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

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

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

2. Aritmética limitada de caracteres

- En C++, los caracteres se pueden manipular como enteros, gracias a su correspondencia con los códigos ASCII. Para realizar la aritmética de caracteres debemos tener en cuenta que el rango de caracteres está entre -128 y 127 o entre 0 y 255 (unsigned char).

```
char ch1 = 125, ch2 = 10;
ch1 = ch1 + ch2; //135
cout << (int)ch1 << endl; // >127
cout << (char)(ch1 - ch2 - 4);
```

```
char c = 'd';
if (c >= 'a' && c <= 'z') {
    c = c - 32;
}
cout << c << endl; // Muestra 'D'
```

- **Ejercicio 1**: Escribir un programa que genere 20 caracteres de forma aleatoria y cuente el número de ocurrencias de cada carácter generado.

Cadenas

- Una cadena es un arreglo (colección) de caracteres (char), terminado en el carácter nulo ‘\0’ (que vale 0) en la memoria.

- Una cadena es un **objeto** cuyo tipo string, está definido en el archivo <string>.

- En lo que sigue nos enfocaremos en cadenas al estilo C, debido a que aún se suelen utilizar en C++. Ejemplos de cadenas:

```
"Buenos Dias" //cadena con 2 palabras
"Juan" //Cadena de una palabra
"" //cadena vacía
```

Los ejemplos anteriores reciben el nombre de **cadena de literales**

Cadenas: Declaración e inicialización

- **Declaración**: Una cadena se declara como un arreglo de char que tenga espacio suficiente para todos los caracteres y el carácter nulo.

```
char s[10]; // almacena hasta 9 caracteres
char t[11]; // almacena hasta 10 caracteres
```

- **Inicialización**: al momento de su declaración.

```
char s [10] = "ejemplo"; // 7 + '\0' y sobran 2
char t[] = "ejemplo "; // arreglo de 8 char
```

En estas formas de inicialización el carácter nulo '\0' es **insertado automáticamente al final** de la cadena.

```
char s [10];// puede contener cualquier cosa
```

Cadenas: Declaración e inicialización

- También podemos inicializar de la siguiente forma:

```
char c[]={'e','j','e','m','p','l','o',' ','1','\0'};
```

✓ Aquí la cadena es inicializada carácter a carácter.
✓ En este método el **carácter nulo debe ser ingresado por el programador**

Initialization	Memory representation
char animal[]="Lion";	
char location[]="New Delhi";	
char serial_no[]="A011";	

Ejemplo: intercambiar el contenido de 2 cadenas

1. **Usando arreglo de caracteres**

Si a y b son arreglos de caracteres y queremos intercambiar sus contenidos, debemos copiarlos de un lado a otro:

```
char a[] = "primero";
char b[] = "segundo";
char t[] ;

copiar (t, a); // copia "primero" a t
copiar (a, b); // copia "segundo" a a
copiar (b, t); // copia "primero" a b
```

```
//Implementar la función copiar
void copiar(char a[], char b[]) {
    //Complete aquí su código para copiar
    //cada carácter de b en a, no olvide
    //terminar con el carater nulo
}

//Intercambiar el contenido de dos
cadenas llamando a la función copiar
```

```
void copiar(char a[], char b[]) {
    int i = 0;
    while(b[i] != '\0'){
        a[i] = b[i];
        i++;
    }
    a[i] = '\0';
}
```

Ejemplo: intercambiar el contenido de 2 cadenas

2. Usando punteros a caracter

Si a y b son punteros a caracteres y queremos intercambiar las cadenas a las que apuntan, podemos realizarlo mucho más rápido:

```
char *a = " primero "

char *b = " segundo ";

char *t;

t = a; // t apunta a primero

a = b; // a apunta a segundo

b = t; // b apunta a primero
```

En este caso, sólo se intercambian los valores de los punteros.

```
//Ejercicio implemente su función intercambio
//para cadenas de caracteres usando punteros
```

```
void intercambio(const char **a, const char **b) {
     const char *t = *a; // t apunta a primero
     *a = *b; // a apunta a segundo
     *b = t; // b apunta a primero
}


int main(){
    const char *a = " primero ";
    const char *b = " segundo ";

    //llamar a la funcion intercambio
    intercambio(&a,&b);


    cout << "a = " << a <<endl;
    cout << "b = " << b << endl;
    return 0;
}
```

Arreglo de caracteres de 2 dimensiones

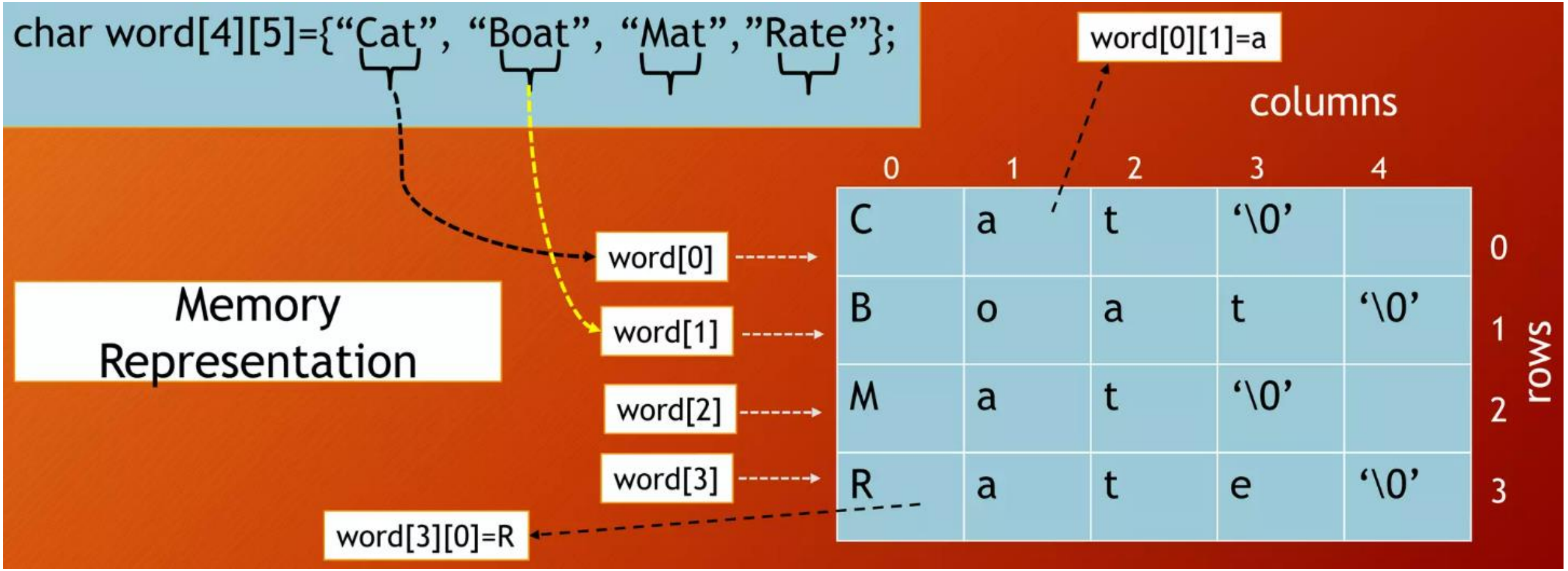
- De la misma forma que interpretamos un arreglo bidimensional, un arreglo de caracteres de dos dimensiones es un arreglo de cadenas (arreglo de arreglo de caracteres)

- Declaración: char lista_alumnos[15][20];
  - ✓ 15 es número de filas (alumnos) representa el número de cadenas.
  - ✓ 20 es el número de columnas, representa el tamaño de cada cadena (cada nombre puede contener hasta 19 caracteres)

- Inicialización: char word[4][5] = {“Cat”, “Boat”, “Mat”, “Rate”};

- Acceso a sus elementos: cout << word[2][0]; imprime M
  cout << word[0]; imprime Cat

Representación en memoria



Ejemplo: Arreglo de cadenas

- Declarar un arreglo de cadenas que contenga los nombres de los 7 días de la semana.

```
char dias[7][10] = {"lunes", "martes", "miercoles",
"jueves", "viernes", "sabado", "domingo"};
```

- Como arreglo adoptaría la siguiente forma.

```
dias[0] = "lunes"; dias[1] = "martes";
dias[2] = "miercoles"; ... ; dias[6] = "domingo";
```

- En la memoria se vería así:

```
"lunes0 .... martes0 ... miercoles0jueves0 ... viernes0 .. sabado0
...domingo0 .."
```

Arreglo de cadenas: Inconvenientes

- En un arreglo de este tipo, todas las cadenas ocupan la misma cantidad de caracteres. Esto quiere decir que cada una debe ser al menos tan larga como la cadena más larga. Por otro lado, las cadenas cortas desperdiciarán mucho espacio.

- Con frecuencia se desea mover una cadena de un lugar a otro en un arreglo (por ejemplo, para hacer copias o mantenerlas ordenadas).

- En este caso se deberá procesar cada carácter de la cadena y entre más largas será más lento.

4. Arreglo de Punteros a caracteres

- Una alternativa es declarar un arreglo de punteros a caracteres. Considerando el ejemplo anterior podría declararse así:

```
char *dias[7] = {"lunes", "martes", "miercoles",
"jueves", "viernes", "sabado", "domingo"};
```

- Esto se interpretaría exactamente igual que un arreglo:

```
dias[0] = "lunes"; dias[1] = "martes";
dias[2] = "miercoles"; ... ; dias[6] = "domingo";
```

- Pero en la memoria se vería distinto: `dias[0]` sería un puntero a la cadena "lunes" que ocupa sólo seis caracteres, etc.

5. Funciones de entrada de datos

- Una Para poder utilizar las funciones que controlan la entrada y salida estándar en C++ es necesario incluir la librería para flujos de entrada y salida de datos en C++: iostream y el espacio de nombres de la librería: std

- Salida estándar en C++:
  La función cout junto con el operador de inserción << permite imprimir cadenas de texto y valores almacenados en variables.

- Entrada estándar en C++:
  La función cin junto con el operador de extracción >>, permite capturar cualquier tipo de datos, con excepción de cadenas de texto que contengan espacios. Mientras que getline() permite la captura de textos con espacios.

Lectura (input) y escritura(output) de cadenas

- **Lectura**: podemos ingresar una cadena de 2 formas:
  1. Usando **cin >>**
  2. Usando las función **`getline()`**

- **Escritura**: Una cadena puede ser mostrada utilizando **cout <<**

- ¿Por qué no necesitamos un bucle para ingresar una cadena?
  Porque **toda cadena tiene un delimitador**, el carácter nulo '\0'

- cin >> ingresa una cadena sin espacios, debido a que el operador de extracción >> detiene la entrada cuando encuentra un espacio en blanco

Lectura de cadenas: getline()

- La función getline() puede ser utilizada **para ingresar texto de varias líneas**

- **Sintaxis**: cin.getline(**cadena**, **MAX**, **Caracter delimitador**), donde:
  - ✓**cadena** es el arreglo de caracteres. Es la variable en la que se almacena la lectura.
  - ✓**MAX** su valor determina el máximo número de caracteres permitidos durante la lectura.
  - ✓**Carácter delimitador**: Es el carácter que, al ser encontrado en el proceso de lectura, detiene la entrada. El carácter delimitador por default es '\n'.

- La función getline continua con la lectura de la cadena hasta que el número máximo de caracteres permitidos es introducido o hasta encontrar el carácter delimitador.

6. Librería de funciones para cadenas

- Requiere el archivo de cabecera <cstring>

- Las funciones toman como argumentos una o más cadenas. Los argumentos pueden ser:
  - ✓ Nombre del arreglo de caracteres
  - ✓ Puntero a char
  - ✓ Literal de cadenas

- Podemos dividirlo en tres grupos:
  1. Para trabajar con cadenas: Requiere el archivo de cabecera <cstring>
  2. Para trabajar con caracteres: Requiere el archivo de cabecera <cctype>
  3. Para convertir datos numéricos y cadenas: require el archivo de cabecera <cstdlib>

6.1 funciones para cadenas

Agregar la librería de cabecera:

```
#include <cstring>
```

Para más detalles de otras funciones en la librería, visitar: https://cplusplus.com/reference/cstring/

Función	Descripción
`strlen(s)`	Retorna la longitud (número de caracteres no nulos) de `c`.
`strcpy(s1, s2)`	Modifica `s1` reemplazando sus caracteres por copias de los caracteres de `s2`.
`strncpy(s1, s2, n)`	Modifica `s1` reemplazando sus caracteres por los `n` primeros caracteres de `s2`. Si `strlen(s2)` es menor que `n`, entonces `s1` es rellenado con 'ceros' hasta que un total de `n` hayan sido escritos en él.
`strcat(s1, s2)`	Modifica `s1` concatenando `s2` al final de `s1`.
`strncat(s1, s2, n)`	Modifica `s1` concatenando con los `n` primeros caracteres de `s2`. Si `strlen(s2)` es menor que `n`, entonces `s1` es rellenado con 'ceros' hasta que un total de `n` hayan sido concatenados al final de él.
`strcmp(s1, s2)`	Compara `s1` y `s2`, retornando un entero negativo, 0, o un entero positivo según `s1` es menor que, igual a, o mayor que `s2`.
`strstr(s1, s2)`	Busca en `s1` por la primera ocurrencia de `s2`, retornando un puntero al primer caracter de esta ocurrencia o `nullptr` si `s2` no es encontrado en `s1`.

6.2 funciones para el manejo de caracteres

Agregar la libreria de cabecera: `#include <cctype>`

Función	Descripción
`islower(ch)`	Retorna `true` si `ch` es minúscula y `false` caso contrario.
`isupper(ch)`	Retorna `true` si `ch` es mayúscula y `false` caso contrario.
`tolower(ch)`	Retorna la minúscula equivalente a `ch` si `ch` es mayúscula.
`toupper(ch)`	Retorna la mayúscula equivalente a `ch` si `ch` es minúscula.

Existen otras funciones tales como: `isalpha(ch)` `isdigit(ch)` `isalnum(ch)` `isspace(ch)` …

Para mayor detalle de funciones en la librería visitar:
https://cplusplus.com/reference/cctype/

6.3 funciones para conversión de datos numéricos

Agregar la libreria de cabecera: `#include <cstdlib>`

Función	Descripción
`atoi(s)`	Convierte `s` a un `int` (si es posible) y retorna este valor.
`atof(s)`	Convierte `s` a un `double` (si es posible) y retorna este valor.
`atol(s)`	Convierte `s` a un `long int` (si es posible) y retorna este valor.

Si s no contiene dígitos, **los resultados no están definidos**:

- La función puede devolver el resultado de la conversión hasta el primer dígito que no sea un dígito
- La función puede devolver 0

Para mayor detalle de funciones en la librería visitar:

https://cplusplus.com/reference/cstdlib/

Ejemplo

Escribir una función que permita convertir una cadena numérica literal a decimal. Utilice arreglo de caracteres, luego puntero a carácter.

```
//Con arreglos
int decimal ( char s[]) {
    int n = 0;
    for ( int i = 0; '0' <= s[i] && s[i] <= '9'; i++)
        n = 10*n + (s[i] - '0');
    return;
}
```

Ejercicio: Escribir la versión con punteros y utilizando funciones de la librería de cadenas

Tokenización de cadenas (separar por delimitadores)

strtok divide una cadena en partes (tokens) utilizando delimitadores como espacios, comas, puntos, etc.

```
char* strtok(char* str, const char* delimitadores);
```

Ejemplo:

```
char texto[] = "uno,dos,tres,cuatro";
char* token = strtok(texto, ",");

while (token != nullptr) {
    cout << token << endl;
    token = strtok(nullptr, ",");
}
```

Más sobre cadenas

La clase **string** ofrece varias ventajas sobre las cadenas al estilo C:

Gran cuerpo de funciones miembro

Operadores sobrecargados para simplificar expresiones

Es necesario incluir el archivo de encabezado `#include <string>`

Para mayor detalle de la librería, visitar:
https://cplusplus.com/reference/string/string/