Lenguaje C

Archivos .h y .c

Librerías

Una librería es un conjunto de código que se puede usar para realizar software. Nos permiten reutilizar funciones y/o usar código hecho por otros.

Por un lado están los usuarios de la librería, que no necesariamente están interesados en modificarla o saber cómo funciona. Quieren saber cómo se usa y que les sea sencillo hacerlo.

Por otro lado están los programadores de la librería que quieren hacerla lo más fácil de usar posible y van a modificar su funcionamiento constantemente.

Archivos .h

Los archivos de *headers* .h contienen las firmas de todas las funciones que se pueden usar en la librería, en general acompañados de comentarios detallando qué hacen y sus casos borde.

```
/*
* Dado el volumen de un depósito D y el volumen de un producto V
* la función calcula cuántos productos entran en el depósito.
*/
int storage_capacity(float d, float v);
```

Archivos .c

Los archivos .c contienen el código de cada una de las funciones que están en el .h además de otras que puedan usarse para implementar esas funciones pero que no estén disponibles para el usuario.

No todo el código en el .c es accesible al usuario.

Macros

Macros

Una macro es un fragmento de *código* al que le damos nombre. El preprocesador va a cambiar las apariciones de nuestra macro por el código correspondiente.

```
#define BUFFER_SIZE 1024

…

char* buffer = malloc(BUFFER_SIZE);
```

stdbool.h

```
#ifndef STDBOOL_H
#define STDBOOL_h

#define bool _Bool
#ifndef true
#define true 1
#endif

#ifndef false
#define false 0
#endif

#endif
```

Funciones con macros

Si definimos:

```
#define SUM(a,b) (a + b)
```

Cuando corremos:

```
c = SUM(x,y);
c = d * SUM(x,y);
```

El preprocesador lo deja de la siguiente forma:

```
c = (x + y);
c = d * (x + y);
```

Strings

Strings

En C las strings son un conjunto de chars, cada char es un número de 0 a 255 que simboliza un carácter de la tabla ascii.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

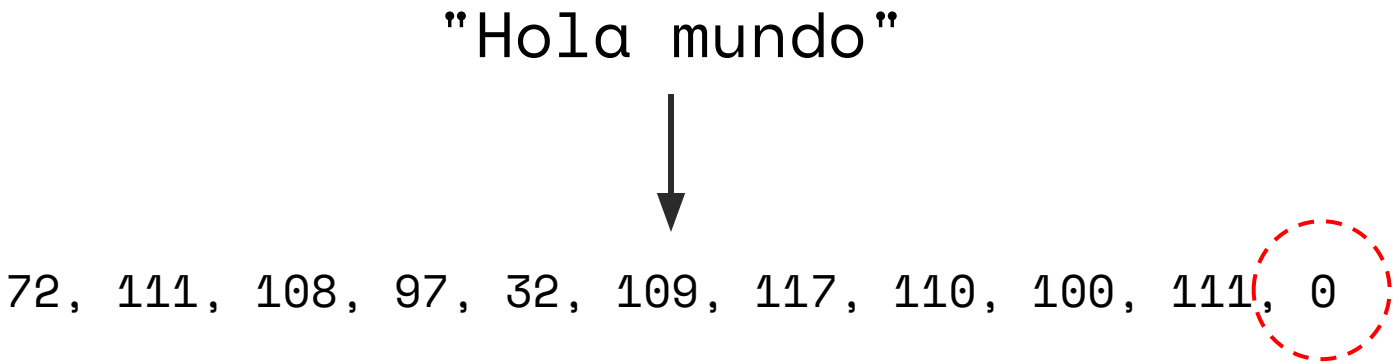
Strings

En C las strings son un conjunto de chars, cada char es un número de 0 a 255 que simboliza un carácter de la tabla ascii.

"Hola mundo"

72, 111, 108, 97, 32, 109, 117, 110, 100, 111, 0

Strings



Se indica con un 0 que el texto terminó, de esta forma no tiene que viajar el largo con nuestra string.

Inicializar Strings en C - Memoria Estática

```c
char str[5] = "Hola";
```

Declaramos un String en memoria **estática** como un **arreglo de char.**

Notar que pedimos **un char extra** para almacenar el carácter nulo. Este es almacenado automáticamente por el compilador.

Inicializar Strings en C - Memoria Dinámica

```c
int size = 2;
char* str = (char*) malloc(sizeof(char) * (size+1));

str[0] = 'o';
str[1] = 'k';
str[2] = '\0';
```

Declaramos el String “ok” en memoria **dinámica** usando malloc.

Notar que reservamos **size+1** bytes para almacenar el carácter nulo.

Calcular el largo de un string

```
size t largo string(const char* string){
    size t i = 0;
    while(string[i] != 0){
        i++;
    }
    return i;
}
```

string.h

La librería string.h contiene funciones para manipular strings, entre ellas:

- `size_t strlen(const char *s)`
  - Devuelve el largo de un string (**Sin contar el char nulo)**
- `int strcmp(const char *s1, const char *s2)`
  - Devuelve 0 si las strings son iguales, distinto de 0 en caso contrario.
- `char *strcpy(char *restrict dest, const char *src)`
  - Copia a s2 en s1
- `char *strcat(char *restrict dest, const char *restrict src)`
  - Añade s2 al final de s1

string.h - the n functions

Pero hay otras *ligeramente* distintas. Tienen una `n` en el nombre:

- `size_t strnlen(const char *s, size_t maxlen)`
  - Devuelve el largo de un string de largo fijo.

Pensemos que pasa si le pasamos un string de 32GB a la siguiente línea:

```
if (strlen(input) > 256) return error_code;
```

O si no tenemos un `\0` en nuestro string.

Ejercicio

Escribir un programa que dado un string imprima el número de palabras que tiene y las palabras.

```
#include <stdio.h>
#include <stdlib.h>

void ejercicio(const char *buffer, unsigned int len);

int main(int argc, char **argv) {
  char *buffer = NULL;
  size_t len;
  int read = getline(&buffer, &len, stdin);
  if (read != -1) {
    ejercicio(buffer, len);
  } else {
    printf("No se leyó ninguna línea\n");
  }

  free(buffer);
  return 0;
}
```

Structs

Structures

Los *structs* o *structures* en C son una forma de agrupar distintos tipos de variables.

```
typedef struct nodo {
    void* dato;
    struct nodo* sig;
} nodo_t;
```

Structures

Para acceder al miembro de una estructura usamos un punto (.)

```
nodo1.sig = nodo2;
```

Si lo que tenemos es un puntero a un struct debemos desreferenciar y usar después el punto.

```
(*nodo1).sig = nodo2;
```

O lo que es lo mismo, usar el operador flecha

```
nodo1->sig = nodo2;
```

Ejercicio

Ejercicio 4.1

```
1  typedef struct Persona {
2      char* nombre;
3      char* apellido;
4      char* domicilio;
5      int edad;
6  } Persona;
7
8  char* masGrande(Persona** personas, int n)
9  {
10     // COMPLETAR
11 }
```

Completar la función *masGrande*, que dado un arreglo de n personas, devuelve el nombre de la persona de mayor edad.

Ejercicio

¿Y si tiene que devolver el nombre completo?

Ejercicio del TAP 2019