

Persistencia de datos

Sofía Beatriz Pérez
Daniel Agustín Rosso

`sperez@iua.edu.ar`

`drosso@iua.edu.ar`

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Clase número 8 - Ciclo lectivo 2022

Agenda

Secuencias y archivos

El puntero a archivo

Escritura y lectura de caracteres

Escritura y lectura de cadenas

Eliminación de archivos

Leyendo y escribiendo datos de más de un byte

Disclaimer

Los siguientes slides tienen el objetivo de dar soporte al dictado de la asignatura. De ninguna manera pueden sustituir los apuntes tomados en clases y/o la asistencia a las mismas.

Es importante mencionar que todos este material se encuentra en un proceso de mejora continua.

Si encuentra bugs, errores de ortografía o redacción, por favor repórtelo a sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: [▶ infoI_IUA_GitLab](#)

Introducción

Almacenar información en archivos es particularmente útil cuando se desea tener *persistencia en los datos*. Es decir, cuando se desea recuperar la información más adelante en el tiempo o bien, transferirla a otra PC.

La ruta de un archivo es la notación en forma de texto (string) que nos permite identificar a un archivo en un dispositivo de almacenamiento:

- Ruta absoluta
- Ruta relativa
- Permisos de usuarios en linux

Secuencias y archivos I

El sistema de entrada/salida de C suministra al desarrollador una interfaz consistente e independiente del dispositivo al que se este accediendo. A esta abstracción se la denomina **secuencia**, mientras que al dispositivo real **archivo**.

- Secuencias: El sistema de archivos de C está diseñado para trabajar con una amplia variedad de dispositivos: terminales, unidades de disco, UARTS, teclados, impresoras, etc. Aunque todos ellos lucen muy distinto, el sistema de archivos de C los transforma en un dispositivo lógico llamado **secuencia**.
 - Secuencias de texto
 - Secuencias binarias

Secuencias y archivos II

- Archivos: Mediante una operación de apertura, se vincula a una secuencia con un archivo específico. Una vez que el archivo está abierto, se puede intercambiar información entre este y el programa.
Por el contrario, se puede desasociar un archivo de una secuencia, mediante la función de cierre.

El puntero a archivo

Un *puntero a archivo* es un puntero a una estructura de tipo **FILE**. Apunta a información que define varias cosas sobre el mismo, incluyendo su nombre, su estado y la posición dentro de el, etc. **Para leer o escribir sobre archivos, los programas tienen que utilizar punteros.**

Para disponer de una variable puntero a archivo, se utiliza una instrucción como la que sigue:

```
1 FILE *fp ;
```

Apertura de un archivo I

La función `fopen()` abre una secuencia para ser utilizada y vincula un archivo con la misma. Después devuelve un puntero al archivo asociado. En general, este archivo es un archivo de disco.

Para disponer de una variable puntero a archivo, se utiliza una instrucción como la que sigue:

1 **FILE * fopen(const char *nombre, const char *modo)**

donde `nombre` es un puntero a una cadena de caracteres que representa un nombre válido de archivo (usualmente la ruta).

Apertura de un archivo II

La cadena que apunta a modo, determina como se abre el archivo.
Algunas de las opciones se listan en la siguiente tabla:

Modo	Significado
r	Abre un archivo de texto para lectura
w	Crea un archivo de texto para escritura
a	Abre un archivo de texto para añadir información
r+	Abre un archivo de texto para lectura y escritura
w+	Abre un archivo de texto para lectura y escritura
a+	Abre un archivo de texto para lectura y escritura para añadir información



Apertura de un archivo III

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     FILE *fp;
6     fp=fopen( " ./ prueba.txt" , "w" );
7     if (fp==NULL)
8     {
9         printf("Imposible abrir el archivo");
10        exit(1);
11    }
12
13    return (0);
14 }
```

*Correr ejemplo con chmod 000 y explicar brevemente los permisos.

Apertura de un archivo IV

- Si se abre un archivo para operaciones de lectura, pero el archivo no existe, la función `fopen()` falla.
- Si se abre un archivo para añadir información, pero el archivo no existe, entonces se crea. Por el contrario, si el archivo existe, la nueva información se agregará al final
- Si se abre un archivo en modo escritura, pero el archivo no existe, entonces se crea
- Si se abre un archivo en modo escritura, pero el archivo si existe, **se borra el contenido y se crea un archivo nuevo**
- La diferencia entre los modos `r+` y `w+` es que con `r+` no se crea el archivo en caso de que no exista; sin embargo, con `w+` si. Si el archivo existe, `r+` no destruye el contenido, `w+` si.

Cierre de un archivo

La función `fclose()` cierra una secuencia que haya sido abierta con una llamada a `fopen()`. Escribe en el archivo toda la información que todavía se encuentre en el buffer del disco y realiza un cierre formal en el archivo a nivel de sistema operativo.

El prototipo de la función `fclose()`:

```
1 int fclose(FILE *fp);
```

Donde `fp` es el puntero al archivo devuelto por la llamada `fopen()`. Si la función `fclose()` devuelve un cero, significa que la operación de cierre ha resultado exitosa.

Escritura de un caracter l

La funciones `putc()/fputc()` escriben un caracter (un entero sin signo), en un archivo que haya sido previamente abierto para operaciones de escritura con la función `fopen()`.

1 `int fputc(int c, FILE *fp);`

Por razones históricas, el caracter a escribir se define como `int`, pero sólo se toma el byte menos significativo (ver tabla). Si la operación de `putc` fue exitosa, esta función retorna el caracter escrito. Caso contrario retorna **EOF**.

Escritura de un caracter II

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	[ENG 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]

Lectura de un caracter

La funciones `getc()/fgetc()` lee caracteres desde un archivo abierto en modo lectura con `fopen()`.

1 `int fgetc(FILE *fp);`

Aunque estas funciones retornan **EOF** cuando se alcanza el final del archivo, se recomienda utilizar la función `feof()` para realizar esta tarea. Esta función retorna un cero cuando se llegó al final del archivo

Ejemplo: loopback lectura y escritura de caracteres I

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int ii = 0;
7      char data='0';
8
9      FILE *fp = fopen("./prueba.txt","w");
10     if (fp == NULL)
11     {
12         printf("Imposible abrir el archivo");
13         exit(1);
14     }
15
```


Ejemplo: loopback lectura y escritura de caracteres II

```
16  do
17  {
18      scanf("%c",&data);
19      fputc(data, fp);
20  } while (data != '$');
21
22  fclose(fp);
23
24  fp = fopen("./prueba.txt", "r");
25  if (fp == NULL)
26  {
27      printf("Imposible abrir el archivo");
28      exit(1);
29  }
30
```



Ejemplo: loopback lectura y escritura de caracteres III

```
31 while (!feof(fp))  
32 {  
33     data=getc(fp);  
34     printf("%c",data);  
35 }  
36  
37  
38 fclose(fp);  
39 return 0;  
40 }
```

Escritura y lectura de cadenas I

Las funciones `fgets()` y `fputs()` se comportan igual que `getc()` y `fgetc()`, sólo que permiten trabajar un sólo carácter, leen o escriben cadenas.

```
1 int fputs(const char *string , FILE *fp );  
2 char *fgets(char *string , int longitud , FILE *fp );
```

La función `fputs()` escribe la cadena apuntada por `string` en la secuencia especificada. Si se produce algún error, devuelve EOF.

La función `fgets()` lee una cadena de la secuencia especificada hasta que llega a un carácter de salto de línea o se hayan leído *longitud* – 1 caracteres.

La función devuelve `string` si se ejecutó correctamente

Escritura y lectura de cadenas II

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main ()
5 {
6     FILE *fp = fopen("./prueba.txt","w");
7     char cadena[46]="Cadena en un array";
8     if (fp == NULL)
9     {
10         printf("Imposible abrir el archivo");
11         exit(1);
12     }
13
14     fputs("Cadena I", fp);
15     fputs("Informatica II - IUA", fp);
```

Escritura y lectura de cadenas III

```
16 fputs(cadena, fp);  
17 fclose(fp);  
18  
19 fp = fopen("./prueba.txt", "r");  
20 if (fp == NULL)  
21 {  
22     printf("Imposible abrir el archivo");  
23     exit(1);  
24 }  
25 fgets(cadena, (46+1), fp);  
26 printf("Cadena recuperada\n%s\n", cadena);  
27 fclose(fp);  
28 return(0);  
29 }
```

Eliminación de archivos

La función `remove()` es utilizada para remover archivos. Recibe el path al archivo a eliminar y devuelve cero si ha tenido éxito en el proceso.

```
1 int remove (const char *nombre);
```

Leyendo y escribiendo datos de más de un byte I

Para leer y escribir datos que ocupen más de un byte, el sistema de archivos de C proporciona dos funciones: `fread()` y `fwrite()`. Estas funciones permiten la lectura y escritura de bloques de cualquier tipo de datos.

```
1 size_t fread(void *buf, size_t n_b, size_t cta, FILE *fp);  
2 size_t fwrite(void *buf, size_t n_b, size_t cta, FILE *fp)
```

- `buf`: es un puntero a la memoria que recibirá la información en caso de una lectura o la que se desea escribir, en caso de una escritura.
- `cta`: determina cuantos elementos se van a leer o escribir
- `n_b`: tamaño en bytes de cada uno de los elementos a leer o escribir

Leyendo y escribiendo datos de más de un byte II

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct measurement
5  {
6      float temp;
7      float hum;
8      int id;
9  };
10
11 #define TAM 3
12
13 void check_null(FILE *fp);
14
15
```


Leyendo y escribiendo datos de más de un byte III

```
16
17 int main (void)
18 {
19     struct measurement m_array[TAM];
20     int ii;
21     FILE *fp;
22
23
24     fp = fopen ("measurement_db.txt", "w");
25     check_null(fp);
26
27
28
29
30
```

Leyendo y escribiendo datos de más de un byte IV

```
31  for ( ii=0; ii<TAM; ii++)
32  {
33      printf ( " Medicion nro: %d\n" , ii );
34      scanf ("%f",&m_array [ ii ].temp );
35      scanf ("%f",&m_array [ ii ].hum );
36      m_array [ ii ].id=ii ;
37      fwrite (&m_array [ ii ] , sizeof (m_array [ ii ]), 1 , fp );
38  }
39  fclose ( fp );
40
41  fp = fopen ( "measurement_db.txt" , "r" );
42  check_null ( fp );
43
44
45
```

Leyendo y escribiendo datos de más de un byte V

```
46     for ( ii=0; ii<TAM; ii++)
47     {
48         fread (&m_array[ ii ],
49             sizeof (struct measurement) ,1,fp );
50     }
51
52     fclose (fp);
53
54     for(int ii=0;ii <3;ii++)
55     {
56         printf(" Temperatura \t %f\n" ,m_array[ ii ].temp);
57         printf(" Humedad \t %f\n" ,m_array[ ii ].hum);
58         printf(" Id \t \t %d\n" ,m_array[ ii ].id );
59     }
60     return (0);}
```

Leyendo y escribiendo datos de más de un byte VI

```
61
62 void check_null(FILE *fp)
63 {
64     if (fp == NULL)
65     {
66         printf("Imposible abrir el archivo");
67         exit(1);
68     }
69 }
```

Las funciones `fprintf()` y `fscanf()` I

Estas funciones se comportan exactamente igual que las funciones `printf()` y `scanf()` estudiadas durante estos cursos. La única diferencia, como es esperable, es que estas operan con archivos.

```
1 int fprintf(FILE *fp, const char *str);  
2 int fscanf(FILE *fp, const char *str);
```



Las funciones fprintf() y fscanf() II

```
1  #include <stdio.h>
2  int main()
3  {
4      FILE *fptr;
5      int id;
6      int ii;
7      char nombre[30];
8      float sueldo;
9      fptr = fopen("database.txt", "w+");
10     if (fptr == NULL)
11     {
12         printf("El archivo no existe \n");
13         return(1);
14     }
15 }
```

Las funciones fprintf() y fscanf() III

```
16  for ( ii=0; ii <3; ii++)  
17  {  
18      scanf ("%d" , &id );  
19      scanf ("%s" , nombre );  
20      scanf ("%f" , &sueldo );  
21      fprintf (fptr , "%d %s %f\n" , id , nombre , sueldo );  
22  }  
23  
24  fclose ( fptr );  
25  return (0);  
26  }
```



Las funciones fprintf() y fscanf() IV

```
1  #include <stdio.h>
2  int main()
3  {
4      FILE *fptr;
5      int id;
6      int ii;
7      char nombre[30];
8      float sueldo;
9      fptr = fopen("database.txt", "r");
10     if (fptr == NULL)
11     {
12         printf("El archivo no existe \n");
13         return (1);
14     }
15 }
```




Las funciones `fprintf()` y `fscanf()` V

```
16 while (!feof ( fptr ))
17 {
18     fscanf ( fptr , "%d %s %f\n" , &id , &nombre , &sueldo );
19     printf ("%d %s %f\n" , id , nombre , sueldo );
20 }
21
22 fclose ( fptr );
23 return ( 0 );
24 }
```

¡Muchas gracias!

Consultas:

`sperez@iua.edu.ar`

`drosso@iua.edu.ar`