Manejo de memoria dinámica parte II
Archivos

Sofía Beatriz Pérez
Daniel Agustín Rosso

sperez@iua.edu.ar
drosso@iua.edu.ar

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Informática II - Clase número 4 - Ciclo lectivo 2023

Agenda

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

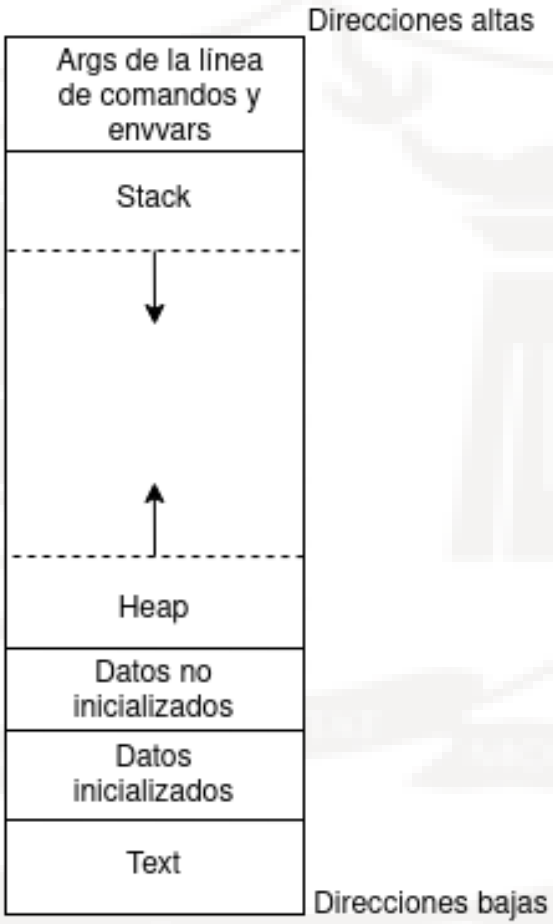
Distribución de la memoria de un programa I

- text o segmento de código: es un segmento de sólo lectura y contiene las instrucciones a ejecutar.
- data: se subdivide en dos partes
  - Datos inicializados (ds): almacena las variables globales y estáticas inicializadas previamente por el desarrollador, antes de la ejecución del programa.
  - Datos no inicializados (bss): almacena las variables globales y estáticas no inicializadas previamente.

Distribución de la memoria de un programa II

- stack: almacena variables locales, argumentos pasados a funciones, y las direcciones de memoria a la que el programa debe retornar luego de la ejecución de una función. (Crece hacia abajo)
- heap: es el segmento donde la asignación dinámica de memoria sucede. (Crece hacia arriba)

Distribución de la memoria de un programa III



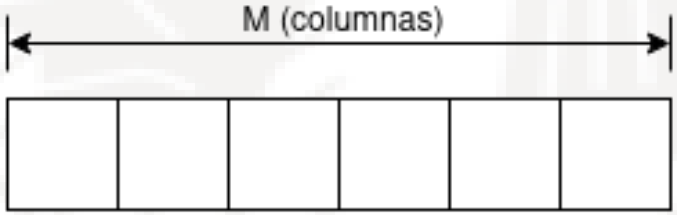
Distribución de la memoria de un programa IV

Analizar en qué sección de layout de un programa estan almacenadas las variables del siguiente programa:

```
1  #include <stdlib.h>
2  int x=0;
3  int y;
4
5  int main(void)
6  {
7      int a = 5;
8      int *p;
9      p = (int *)malloc(sizeof(int));
10     *p = 10;
11     return 0;
12 }
```

Creando arrays dinámicos de más de una dimensión I

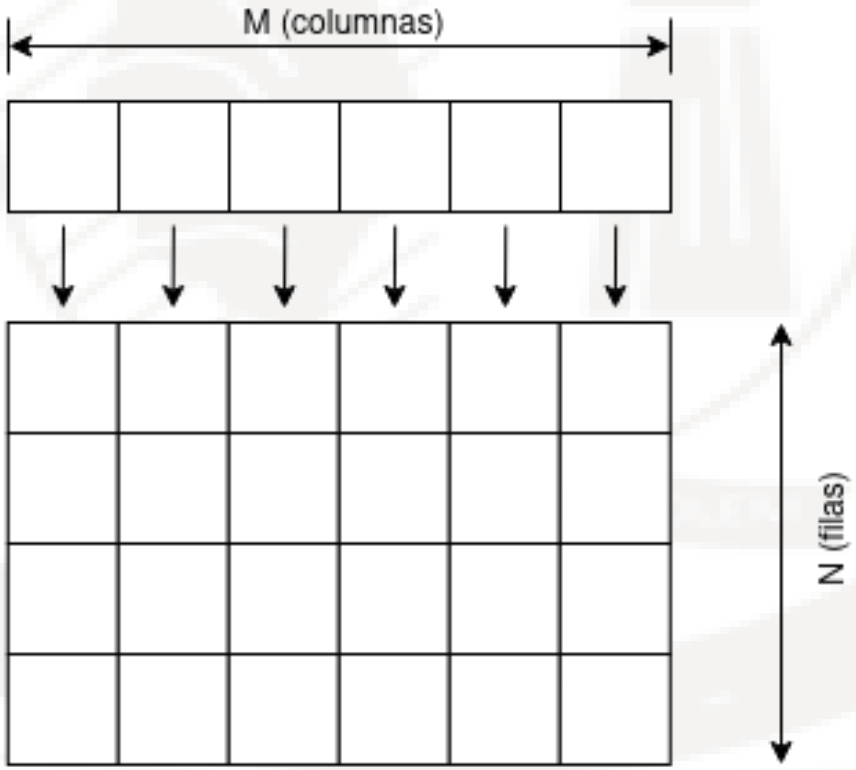
Supongamos que se necesita crear un arreglo de dos dimensiones (M columnas por N filas) de números enteros.
En primer instancia, se creará un arreglo de punteros de tipo int, capaz de almacenar M referencias a columnas:




```
1  int **A = (int **)malloc(M * sizeof(int *));
2  if (A == NULL)
3  { printf("No hay suficiente memoria");
4    exit(0);
5  }
```

Creando arrays dinámicos de más de una dimensión II

Ahora, haremos que cada puntero de M apunte a un espacio de memoria de N elementos:



Creando arrays dinámicos de más de una dimensión III

```
1  for (ii = 0; ii < M; ii++)
2  {
3      A[ii] = (int *)malloc(N * sizeof(int));
4      if (A[ii] == NULL)
5      {
6          printf("No hay suficiente memoria");
7          exit(0);
8      }
9  }
```

Creando arrays dinámicos de más de una dimensión IV

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define M 4
4  #define N 5
5  int main()
6  {
7      int ii=0,jj=0;
8      /*Columnas*/
9      int **A = (int **)malloc(M * sizeof(int *));
10     if (A == NULL)
11     {
12         printf("No hay memoria suficiente");
13         exit(0);
14     }
15
```

Creando arrays dinámicos de más de una dimensión V

```
16      /*Filas*/
17      for (ii = 0; ii < M; ii++)
18      {
19          *(A+ii) = (int *)malloc(N * sizeof(int));
20          if (*(A+ii) == NULL)
21          {
22              printf("No hay memoria suficiente");
23              exit(0);
24          }
25      }
26
27
28
29
30
```

Creando arrays dinámicos de más de una dimensión VI

```
31          /*Carga de datos*/
32          for (ii = 0; ii < M; ii++)
33          {
34              for (jj = 0; jj < N; jj++) {
35                  *(*(A + ii) + jj) = rand() % 100;
36              }
37          }
38          /*Impresion de  datos*/
39          for (ii = 0; ii < M; ii++)
40          {
41              for (jj = 0; jj < N; jj++) {
42                  printf("%d ", *(*(A + ii) + jj));
43              }
44              printf("\n");
45          }
```

Creando arrays dinámicos de más de una dimensión VII

```
46
47          /*Liberacion de filas*/
48          for (ii = 0; ii < M; ii++) {
49              free(*(A + ii));
50          }
51
52          free(A);
53
54          return(0);
55          }
```

Archivos: 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

Archivos: 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

Archivos: 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 intercambia información entre este y el programa.
  Por el contrario, se puede desasociar un archivo de una secuencia, mediante la función de cierre.

Archivos: 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;
```

Archivos: 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).

Archivos: 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

Archivos: apertura de un archivo III

Diseñar y codificar un programa en C que permita abrir un archivo llamado "prueba.txt" en modo escritura.
Modificar los permisos y evaluar la salida de la función fopen().

▸ Ver en git (fopen_example)

Luego analice:

- Sintaxis de la función fopen
- Sintaxis del puntero a archivo
- Salida de la función fopen según los modos de apertura.

Archivos: 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, entonce 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, entonce 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.

Archivos: cierre de un archivo

La función fclose() cierra una secuencia que haya sida 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 cierra 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.

Archivos: escritura de un caracter I

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**.

Archivos: 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

Diseñar y codificar un programa en C que permita abrir un archivo llamado "prueba.txt" en modo escritura. Luego, recibir caracteres ingresados desde el teclado y guardarlos en el archivo mencionado anteriormente. Una vez ingresado el caracter "$", se debe finalizar la toma de datos y cerrar el archivo.
Finalmente abrir el archivo e imprimir su contenido.

▸ Ver en git (put_get_loopback)

Luego analice:

- Sintaxis e implementación de la función fputc()
- Sintaxis e implementación de la función fclose()
- Sintaxis e implementación de la función getc()
- Sintaxis e implementación de la función fclose()
- Codificación ASCII

Archivos: escritura y lectura de cadenas I

La funciones fgets() y fputs() se comportan igual que getc() y fgetc(), sólo que permiten trabajar un sólo caracter, 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 secuecuencia especificada. Si se produce algún error, devuelve EOF.
La función fgets() lee una cadena de la secuencia especificada hasta que llega a un caracter de salto de línea o se hayan leído *longitud* − 1 caracteres.
La función devuelve string si se ejecutó correctamente

Archivos: 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);
```

Archivos: 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);
```

Archivos: 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 fpintf(FILE *fp, const char *str);
2 int fscanf(FILE *fp, const char *str);
```

Archivos: las funciones fprintf() y fscanf() II

Diseñar y codificar un programa en C que permita abrir un archivo llamado "database.txt" en modo escritura. Luego, recibir desde el teclado un número entero asociado al ID, un arreglo de chars asociado al nombre (longitud máx: 30) y un float que represente al sueldo, conformar un string y almacenarlo en el archivo abierto anteriormente. Finalmente abrir el archivo en modo lectura e imprimir su contenido.

▸ Ver en git (set_database y get_database)

Luego analice:

- Sintaxis e implementación de la función fscanf()
- Sintaxis e implementación de la función fscanf()
- Sintaxis e implementación de la función feof()
- Sintaxis e implementación de la función fclose()

Archivos: las funciones fprintf() y fscanf() III

Luego cuando se estudie el tema estructuras, se veran las funciones fread() y fwrite(). Estas funciones permiten la lectura y escritura de bloques de cualquier tipo de datos.

¡Muchas gracias!

Consultas:

sperez@iua.edu.ar
drosso@iua.edu.ar