Obligatorio 3 - Ejercicios con punteros y memoria dinámica

7 de junio de 2018

1. Generalidades

En este obligatorio les pediremos que resuelvan 2 ejercicios similares a los del libro de prácticos, con énfasis en el uso de punteros y de memoria dinámica.

Es importante recalcar que **tanto la prueba escrita como el proyecto entregado son individuales.** El sistema de recepción de entregas, además de almacenar los archivos enviados por los estudiantes, realiza un control de copias contra las entregas de otros estudiantes así como de programas similares que se encuentran en la web. Ese programa es capaz de detectar copias "maquilladas", es decir donde se cambiaron nombres de variables u otras formas de ocultar una copia. Este asunto debe ser bien entendido. No tenemos ningún inconveniente en que discutan soluciones, miren en la web, etc. pero el trabajo entregado debe ser realmente el producto de vuestro trabajo y si el programa de control de copia detecta que hubo copia ello implica una sanción que puede implicar la pérdida del curso e incluso sanciones mayores, tal como está especificado en el reglamento de la Facultad.

En la evaluación se verificará que los los programas compilen y den los resultados esperados.

1.1. Formato del archivo a entregar

El archivo entregado debe ser un archivo comprimido en formato **zip** (NO se aceptan archivos en formato rar), de nombre **nombres_separados_por_infraguiones.apellidos_separados_por_infraguiones.zip** y que los fuentes estén en la raíz del zip. El contenido del archivo debe incluir los siguientes elementos (que deben estar en la raíz del mismo y no en un directorio interno):

- Todos los archivos fuente creados por el estudiante (**.h** y **.c**)
- Un archivo **Makefile** para compilar el o los programas requeridos en el trabajo.

Por ejemplo, supongamos que el obligatorio consiste en la generación de dos ejecutables de nombre `ejercicio1` y `ejercicio2`, su nombre es Juan Pablo Perez Fernandez, y usted implementó dicho ejecutable en los archivos `ejercicio1.c` y `ejercicio2.c`, además de un `Makefile`. Entonces debe subir un archivo de nombre `Juan_Pablo.Perez_Fernandez.zip` con el siguiente contenido:

```
ejercicio1.c
ejercicio2.c
Makefile
```

El **Makefile** en este caso debe ser así:

```
all: ejercicio1 ejercicio2
COPT=-Wall -ansi

ejercicio1: ejercicio1.c
	cc $(COPT) -o $@ ejercicio1.c

ejercicio2: ejercicio2.c
	cc $(COPT) -o $@ ejercicio2.c
```

Nota: Pueden crear un zip desde la máquina virtual con el comando `zip`; la sintaxis es, desde la carpeta de trabajo:

```
$zip -r nombre_archivo.zip .
```

en el ejemplo anterior, sería `zip -r Juan_Pablo.Perez_Fernandez.zip .`

2. Ejercicio 1

Escribir una función de tipo `void` que reciba dos argumentos: un entero (que será utilizado como entrada a la función) y un puntero a puntero a `char` (que será utlizado como salida de la función).

La función será declarada de la siguiente manera

```
void ParImpar(int a, char ** buf)
```

La función debe:

- Evaluar si el entero es par o impar;
- En función de lo anterior debe reservar una zona de memoria de tamaño apropiado (usando la función `malloc`) y poner en ese espacio de memoria reservado la cadena de caracteres correspondiente a las palabras "par" o "impar", respectivamente.
- Devolver en el segundo argumento la dirección de la cadena correspondiente;

En el `main` se deberá:

- Llamar a la función `ParImpar` pasándole como parámetro el primer argumento de la línea de comando. Por ejemplo, si el ejecutable se llama `ejercicio1` y es invocado de la siguiente forma: `./ejercicio1 n`, entonces el main debe llamar a la función `ParImpar` con el valor del entero `n` como primer argumento.
- Según el resultado de la función `ParImpar`, se deberá imprimir un mensaje en pantalla que diga si el valor de `n` es par o impar. El mensaje debe decir: `el número ingresado es par` o `el número ingresado es impar`, según sea el caso.
- Antes de finalizar se debe liberar la memoria reservada, en el `main`. Noten que la reserva de memoria se hizo al interior de la función `ParImpar` pero la liberación se hace en el `main`.

Para probar la función deben correrla y se debe imprimir el mensaje correspondiente al carácter par o impar del parámetro.

Nota: Este ejercicio se ha elaborado solo a los efectos de evaluar ciertas cosas: el uso de punteros y de memoria dinámica, en particular. Ello no quiere decir que sea una buena práctica programar de la manera en que se está solicitando acá.

3. Ejercicio 2

a) Escriba un programa `mem` que reserve un bloque de memoria dinámica para almacenar 1000000 enteros usando `malloc`, luego asigne el valor 0 a todos esos enteros mediante un ciclo `for`, y finalmente libere la memoria utilizando `free`. Mida el tiempo de ejecución utilizando el comando `time` de linux (lea su manual con `man time` para entender la salida):

```
\$ time ./mem
```

b) Escriba un programa como el anterior pero que en lugar de `malloc` y un `for`, utilice la función `calloc` para reservar e inicializar los elementos a 0 automáticamente. Vuelva a medir el tiempo de ejecución.

c) Escriba un programa que reserve 10 enteros con `malloc`, luego les asigne los valores 1 a 10, luego invoque `realloc` para agrandar el espacio reservado a 20, imprima los valores de esos 20 elementos, y finalmente invoque a `free` para liberar el espacio reservado.

d) Supongamos que asigna el bloque de memoria reservado por `realloc` en el inciso *c* a un puntero de nombre `p`. Pruebe asignar un valor a un lugar fuera del array, por ejemplo, `p[-1]` y `p[21]`, a ver qué sucede.

e) Pruebe invocar `free` sobre el mismo puntero dos veces consecutivas en alguno de los programas anteriores.

f) Pruebe invocar `free` sobre un puntero de valor `NULL` en alguno de los programas anteriores.

Haga un pequeño informe comentando cada ejercicio y los resultados obtenidos. En dicho informe nos interesa que expliquen:

- Cual es la diferencia entre malloc y calloc?
- Hay diferencia de tiempo de ejecución entre los incisos *a* y *b*?
- Cual es la diferencia entre hacer un nuevo malloc o hacer realloc?
- Qué pasa si no hacemos free al finalizar el uso de una zona de memoria reservada?
- Qué puede pasar cuando intentamos escribir fuera del rango de la memoria reservada?