Universidad Tecnológica Nacional – Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sistemas Operativos (082027)

GUÍA DE EJERCICIOS DE PROGRAMACIÓN EN C

Sistemas Operativos

UTN-FRBA

Versión 1.0

Objetivos de la Guía

Esta guía de ejercicios tiene como objetivo que el alumno mediante ejercicios teórico-prácticos, adquiera conocimientos sobre algunos temas importantes tanto para la parte teórica como para la parte práctica de la materia.

Al final de esta guía encontrará la resolución propuesta de cada uno de los ejercicios, con una breve explicación en cada caso.

Es importante remarcar que las soluciones propuestas no son las únicas válidas ni tampoco son universales (según el caso concreto de aplicación podría requerir algunos cambios para poder ser utilizada correctamente).

Se recomienda que a medida que el alumno vaya resolviendo esta guía, consulte en el foro, a los docentes, o a los ayudantes, sobre su solución, de forma de detectar algún error, no del lado de la solución propiamente dicho sino de cómo se implementó.

Recomendaciones para bajar y ejecutar los ejercicios y ejemplos

El código fuente de los ejercicios/resoluciones se encontrarán en el siguiente repositorio de GitHub https://github.com/sisoputnfrba/so-exercises.
Para su correcta ejecución debe importarse al workspace sobre el cual estemos trabajando tanto el problema como su resolución por separado de cada ejercicio (para que no existan errores de múltiples funciones `main`).

Temas:

Mis primeros pasos...

Ejercicio 1 (Básico de C)

Realizar las siguientes funciones (revisar qué funciones de Ansi C nos pueden ayudar):

```
/*
*   Retorna un String nuevo que es la concatenación
*   de los dos Strings pasados por parámetro
*   Ejemplo:
*   char* nombre = "Ritchie";
*   char* saludo = string_concat("Hola ", nombre);
*   =>
*   saludo = "Hola Ritchie"
*/
char* string_concat(const char*, const char*);
```

```
/*
*   Asigna en el tercer parámetro, la concatenación
*   de los primeros dos Strings
*   Ejemplo:
*   char* nombre = "Ritchie";
*   char* saludo;
*   string_concat("Hola ", nombre, &saludo);
*   =>
*   saludo = "Hola Ritchie"
*/
void string_concat_dinamyc(const char*, const char*, char**);
```

```
/*
*   Separa el mail en un usuario y un dominio.
*   Ejemplo:
*   char* mail = "ritchie@ansic.com.ar";
*   char* user, dominio;
*   mail_split(mail, &user, &dominio);
*   =>
*   user = "ritchie"
*   dominio = "ansic.com.ar"
*/
void mail_split(const char* mail, char** user, char** dominio);
```

Ejercicio 2 (Modelado de TADs)

Se pide modelar un TAD que represente un Archivo, se debe poder abrir (definir la forma: read/write/append), cerrar un archivo, leer una linea determinada, exponer una operación que reciba una función y la ejecute por cada línea del archivo; exponer otra operación que dada una lista y una función, itere la lista y escriba sobre el archivo abierto lo que

devuelve dicha función (string).

Ejercicio 3 (Uso de Commons-Library)

Dado un archivo de texto que contiene un conjunto de personas, leerlo, procesarlo y generar la salida especificada. Recomendación: Revisar las funciones provistas en la biblioteca ***commons-library*** de la cátedra.

Archivo de entrada:

- Formato => ***Región; Nombre y Apellido; Edad; Número Telefónico; DNI; Saldo***
- Cada renglón representa una persona, cada campo de la persona se encuentra separado por un ';' como delimitador (por lo tanto tienen campos de longitud variable).
- El archivo no sigue ningún orden.

Archivo de salida:

- Formato => **Región | Edad | DNI | Nombre y Apellido (30 chars máximo) | Número telefónico**
- Cada renglón representa una persona.
- El archivo debe estar ordenado por Región y Edad.
- Filtrar los menores de edad (< 18 años).
- Loggear las personas cuyo saldo sea menor a 100$.

Errores comunes: Todos la hemos manqueado alguna vez...

Ejercicio 3

Leo empezó a hacer limpieza de su PC y encontró sus proyectos de 1º año. Empezó a revisar el código fuente y vio cosas que no le parecía que estuvieran bien, pero como él todavía no cursó Sistemas Operativos, no tiene los conocimientos del alumno, por eso se pide que analicen el archivo Errores.c y encuentren los errores que detecten.

Para poder resolver este ejercicio si se utiliza Eclipse es necesario deshabilitar los warnings, ya que sino este nos mostrará sus sugerencias.

*Para esto hay que hacer botón derecho sobre el proyecto y luego ir a **Properties->C/C++ Build->Settings** en la sección **Compiler->Warnings** destildamos show all warnings (**-Wall**) y tildamos inhibit all warnings (**-w**). Esto evitará que nos muestre los warnings que detectarían la mayoría de los errores en el código. Además hay que ir a **Window->Preferences->C/C++->Code Analysis** y destildar todas las warnings.*

Esto debe hacerse antes de agregar los source al proyecto ya que automáticamente el Eclipse analiza y marca las warnings.

Memory Leaks: Eternal Sunshine...

Ejercicio 4

Antes de resolver este ejercicio, te recomendamos leer El Tutorial de Valgrind.

Leo estuvo programando toda la noche para llegar a la entrega de Operativos, codeó como loco para llegar, pero se olvidó de un gran detalle: sus procesos consumen cerca de 500 MB de memoria RAM en plena ejecución.

Leo no entiende mucho lo que pasa con la memoria por eso se le pide al alumno que con la ayuda de ***Valgrind*** arregle todos los memory leaks y errores de memoria varios que se encuentran en el archivo ***Leaks.c*** .

Concurrencia: Competencia sana y no tanto..

Ejercicio 5

Julieta y Leo abrieron una cuenta en el almacén de la esquina, para hacer las compras diarias. Ellos van haciendo las compras cuando necesitan, pero como son muy independientes, no se avisan del último saldo disponible.

Como al almacenero lo estafaron un par de veces, no deja que sus clientes se pasen del saldo que tienen en la cuenta. Leo se propone hacer un pequeño sistema que permita administrar el dinero de la cuenta que tienen en el almacén

Después de estar horas y horas programando, Leo llegó a la solución que se encuentra en el archivo ***SolucionLeo.c***

1. Sin ejecutar esta solución, ¿ve algún problema?
2. Ejecutar la solución y ver si funciona (Leo y Julieta deben poder poder comprar cosas sin que la cuenta del almacén quede con saldo negativo)
3. Ejecutar un par de veces más y prestar atención al orden de ejecución

Lo probaron el primer mes y el almacenero les advirtió que en cierto momento del mes su saldo pasó a ser negativo y que solo los perdonaba por estar tantos años en el barrio, pero que si se repetía esta situación les cerraría la cuenta definitivamente. Por lo tanto, Julieta se puso como loca y ahora le pide al alumno que reformule la solución de Leo, evitando así que les cierren la cuenta.

Modificar la solución para que se asegure la integridad del saldo.
Utilice el mecanismo de sincronización más apropiado.

Ejercicio 6

En la empresa en la que trabaja Leo compraron una impresora nueva de última generación, “All in one” pero, como es normal en estas impresoras, vino con fallas en el sistema de gestión de la cola de impresión.
Esta impresora se usa de forma remota desde 3 PC’s.

Alguna de las características de la impresora son las siguientes:

- La cola tiene un límite de 10 trabajos de impresión
- Puede recibir pedidos al mismo tiempo
- Si la cola de trabajos se termina la impresora debe suspenderse.
- Si la cola de trabajos esta llena la PC que envíe debe esperar a que haya un lugar

Le piden a Leo que desarrolle una solución, la cual se adjunta, pero como esta medio traumado por el error en el sistema del almacenero y no quiere ser despedido, pide al alumno que revise y corrija la solución que ha desarrollado.
Utilice el mecanismo de sincronización más apropiado.

Ejercicio 7

El primo de Leo es gerente en Wendy’s y, luego del furor de la apertura, se dio cuenta de que sus empleados no terminan de entender el circuito productivo de las hamburguesas.

Cada empleado es encargado de agregar un ingrediente, y para ordenarlos se armó un sistema de alertas el cual avisa

cual es el próximo ingrediente a agregar.

La secuencia de pasos para armar las famosas hamburguesas cuadradas son:

- Poner los panes sobre la cinta
- Ponerle condimentos a los panes
- Poner carne
- Poner queso
- Poner panceta
- Poner lechuga
- Poner tomate

Es importante remarcar que la secuencia de pasos debe ser inalterable, de lo contrario podría salir una hamburguesa sin panceta (algo inaceptable para la clientela de hoy en día).
La solución propuesta por Leo se adjunta en el archivo ***Wendys.c***, se le pide al alumno revisarlo y corregirlo de ser necesario.

Serialización: Goku no tenía estos problemas.

Ejercicio 8

Estamos en fecha estelar 85785.9, y el capitán Kirk está muy preocupado últimamente. Envió a Spock a una misión crítica para la seguridad espacial, pero Scotty (el ingeniero de la nave) estuvo haciendo updates al sistema de teletransportación, y Spock nunca llegó a destino

Investigando el sistema Scotty encontró un archivo flatSpock.bin, luego de horas de investigación terminaron descubriendo que la falla se encontraba en el sistema de de-serialización y que ese archivo corresponde a la teletransportación fallida de Spock.

Se le pide al alumno hacer un programa que lea el archivo .bin y lo deserialize correctamente para poder recuperar a Spock.

El capitán Kirk quiere mandar a toda costa a Spock a esa misión, sin embargo, a pesar que la tripulación del Enterprise posee ART, el capitán no quiere volver a arriesgarse, así que ha dejado a su cargo también rehacer el sistema de serialización.

Para que no vuelva a pasar lo mismo, comparar el hash md5 del archivo ***flatSpock.bin***, con el que genere su programa, ambos deberían dar el mismo código.

Se adjunta el sistema original de teletransportación para hacer los cambios que el alumno crea pertinentes y así poder cumplir esta misión.

Estructura de spock en el archivo ***flatSpock.bin***

Campo	Longitud	Tipo
Edad	1 byte	Numérico
Nombre	Variable (n + 1)	String (termina en '\0')
Mascota_Edad	1 byte	Numérico
Mascota_Da_Vueltas	1 byte	Booleano
Mascota_Apodo	Variable (n + 1)	String (termina en '\0')
Mision_Informacion	Variable (n + 1)	String (termina en '\0')
Mision_Longitud	4 bytes	Numérico sin signo
Villanos_Cantidad	4 bytes	Numérico sin signo
Villanos	(n veces * tamaño villano) bytes	
Villano_Nombre	25 bytes	String (termina en '\0')

Villano_Edad	2 bytes	Numérico sin signo

Networking: Conectados somos más...

Ejercicio 9

Leo se va de viaje a Europa, y como es muy pegote con la novia, quiere estar comunicado con Julieta todo el tiempo, para esto se le pide al alumno (que tiene una vasta experiencia en programación con sockets) le haga un sistema de chat.
El programa consistirá de 2 procesos los cuales deben abrir un canal de comunicación a través del cual un iniciará la conexión en modo servidor y otro se conectará como cliente y luego se enviaran mensajes los cuales serán mostrados por pantalla.

Ejercicio 10

Marcela, la madre de Juli, se enteró del sistema que está haciendo el alumno y se puso muy celosa, por eso Leo pide una molificación, el chat debe poder mantenerse entre más de 2 personas, cambiando la estructura del sistema. Ahora tendrá:

- 1 proceso servidor que distribuye los mensajes
- N procesos clientes que se comunicaran con el servidor (probar con 3 procesos, pero el mecanismo debe soportar una cantidad ilimitada de clientes

Resoluciones

Ejercicio 1

No se implementa.

Ejercicio 2

No se implementa.

Ejercicio 3

Error n°1 : recorre de 1 a 10 cuando los arrays van de 0 a elem-1 (0 a 9 en este caso)

Error n°2: cada case debe tener su break, caso contrario se da lo que se llama *fall-through*, una vez que entra en un switch ejecuta todos los case que estén debajo hasta que se encuentre con el final, o con un break.

Error n°3 : los caracteres se representan con comillas simples. En una variable de tipo char se almacena un valor de 0 a 256 que representa el código de ese carácter. Si nosotros asignamos "D" pasan varias cosas. Primero se aloca memoria automáticamente para el String (o char array mejor dicho) "D". Al asignar "D" al char, no se esta asignando la D, sino el puntero que se genero (el cual apunta a D). Por ende le estamos asignando a cadena[i] un valor incoherente (internamente se convertirá a char ese valor, pero SEGURO no será el valor correspondiente a la D).

Error n°4: Usar operaciones de *strings* sobre *streams*. Es importante recalcar la diferencia entre ambos, la cual radica que si bien los dos son una secuencia de bytes, los *strings* terminan en el caracter nulo, representado por el '\0'. Las funciones que esperan *strings* (printf, strcpy, strcmp) operan desde la dirección del puntero que se pase por parámetro hasta el primer '\0' que encuentre. En este caso la variable cadena no tiene '\0' por ende si usamos una función de este tipo, seguirá recorriendo bloques de memoria hasta que encuentre un ´\0´ corriendo el riesgo de corromper la memoria. También hay que tener en cuenta el otro caso, que es que si un *String* tiene un '\0' en la mitad, el uso de alguna de estas funciones no resultará correcto ya que analizara solo hasta el primer '\0'.

Error n°5 : El error acá radica en que la función ***strcmp***, esta devuelve 0 si 2 *strings* son iguales, y el if pregunta siempre por 1 en la expresión-condición. Si esta evalúa a algo distinto de 1, entonces iría por el false.

Error n°6: Esto es un error muy común de los que programan en lenguajes de alto nivel. No se puede comparar cadenas con el ==. Esto sucede primero porque no existe la estructura de string, solo existen punteros, y justamente las 2 variables que tenemos aquí son 2 char*. Al compararlas con == lo que comparamos son los valores de las variables (que son direcciones, no cadenas) y obviamente ambas son diferentes.

Error n°7 : El ***strcpy*** no aloca memoria, el char* de destino debe tener memoria alocada y lo suficientemente grande para almacenar la de origen.

Error n°8: Este error radica en que no se hizo un malloc lo suficientemente grande porque el strlen(..) devuelve la cantidad de caracteres SIN EL CARÁCTER NULO. Cuando copiamos con strcpy(..) copiamos también el '\0',

por eso debe agregarsele un +1 al size del malloc así entra el '\0'.

Error n°9 : Este es un error dado por un concepto, el pasaje por valor o por referencia. En C ***siempre*** se hace pasaje por ***copia*** del valor, no existe el pasaje por referencia.
Cuando hacemos
int a=2;
sqrt(a);
lo que se hace, es que se agarra el valor que se paso por parámetro y se copia al stack de la función llamada. Si yo adentro de sqrt quiero hacerle modificaciones al value (parámetro), podre, pero ***ninguna de esas modificaciones*** afectará a la variable *a*.
Si yo tengo
char* cadena="Hola Mundo";
ModificarCadena(cadena);

y dentro de *ModificarCadena(char* value)* hacemos value="Otra cadena mejor"

Ahí estamos bajo el mismo caso, ya que cadena tiene como valor una dirección, la dirección de comienzo del String. *value*, cuando comienza la ejecución de *ModificarCadena* tiene el mismo valor que cadena. al momento de asignar value ***no estamos modificando*** la variable cadena.
Para poder hacer esto habría que hacer
char* cadena="Hola Mundo";
ModificarCadena(&cadena);

y dentro de *ModificarCadena(char** value)* hacemos *value="Otra cadena mejor"

Error n°10 Y 11: scanf pide en su segundo argumento punteros. En el primer caso esta mal, porque es no es un puntero, para pasar el puntero deberíamos hacer &x. En el segundo caso también esta mal, porque *st es el contenido del puntero (la cadena) y no el puntero, en este caso con pasar st alcanza.

Error 12: El error aquí, es con los prototipos de las funciones. El compilador asume que todas las funciones que no tengan encabezados devuelven un int, y reciben una cantidad ilimitada de argumentos de cualquier tipo. En el caso de la función sqrt, que devuelve un double, esto genera errores, ya que el retorno de la función no sera 1.4..... sino 1.

Error 13: Es un error parecido al 9, para modificar la variable raíz (y poder elevarla al cuadrado) se debe pasar el puntero a la variable, no su valor.

Error 14: Es un error común, el operador de comparación es == , no =. Lo peor de esto es que solo genera un warning. ¿Por qué? porque es válido hacer una asignación en una condición. Es importante recordar que el resultado de una expresión asignación, es el valor asignado, es decir que si hago a=2; esta expresión evalúa a 2.
Dicho esto, en el if una asignación funciona igual

if(a=2)->if(2)->ELSE porque es distinto de 1.

Error 15: no es un error sintáctico. Acá por un error de indentación del código se imprime el mensaje equivocado. Recordar que un else siempre aplica al if más cercano.

Ejercicio 4

El archivo contiene los siguientes errores:
- Archivo ***Leaks.c***
    * Cuando se aloca espacio para la variable ***absolute_path*** falta agregarle un byte por el '\0' que lleva toda palabra.
- Archivo ***Cliente.c***
    * Cuando se destruye el cliente falta liberar la memoria que tiene el valor del nombre y del apellido.
    * Se utiliza una variable automática para que el cliente apunte a dicho valor, el problema es que una vez que termina la ejecución de la función la posición de memoria de dicha variable puede ser utilizada para cualquier otra cosa y esto puede corromper el valor del nombre y del apellido.
- Archivo ***Factura.c***
    * Falta destruir los elementos que están referenciados por el TAD Lista.

Ejercicio 5

La solución de Leo no es correcta, ya que no esta correctamente sincronizada. Como pueden haber visto, el orden de ejecución varía en cada una de las ejecuciones, esto se debe a que el orden en el que se ejecuten los hilos depende de la planificación del SO, nosotros (los programadores) no podemos asumir nada sobre el orden de ejecución de los hilos.
Más allá del orden en el que se ejecuten los hilos, el problema es que no se está garantizando la mutua exclusión al acceder a la variable ***saldo*** la cual es utilizada tanto para consulta como para escritura en la función ***hacer_compras(..)***.
Para garantizar esto tenemos 2 opciones, utilizar un Mutex o un semáforo binario.
El mejor mecanismo de sincronización en este caso es un Mutex ya que no importa el orden de ejecución solo importa que haya Mutua Exclusión.

Ejercicio 6

Esta solución de Leo tiene 2 problemas:
    * No se garantiza la Mutua Exclusión en el acceso a la cola, tanto para agregar como para quitar elementos.
Para resolver este inconveniente lo que hacemos es agregar un Mutex que asegurará la Mutua Exclusión al agregar o quitar elementos de la cola.

    * No se cumple con la definición funcional que pide que la impresora debe suspenderse si no hay más trabajos disponibles para procesar, y que si luego aparecen debe accionarse sola.
Podemos ver en el código que si no hay trabajos sale del while y el hilo finaliza su ejecución. Reemplazar ese while por un while(true) tampoco es una opción, ya que si siempre se trata de hacer un pop de la cola y esta está vacía, generará una espera activa consumiendo el uso del CPU con procesamiento innecesario.
Para resolver este inconveniente lo que hacemos es agregar dos semáforos contadores, uno se utiliza para que solo se active cuando hay trabajos pendientes en la cola de impresión y el otro se utiliza para bloquear al hilo que produce cuando se llego al limite de trabajos en espera.

Ejercicio 7

El problema de la solución propuesta es que Leo pensó que el orden de creación de los hilos eran el orden en el que se iban a ejecutar, pero así como venimos viendo con ejercicios anteriores, sabemos que esto no es así. Los hilos se planifican de acuerdo a la disponibilidad y algoritmo del SO y por ende no se puede asumir que el

orden de creación de los hilos definirá el orden de la ejecución. Para sincronizar correctamente acá hay una única opción, un semáforo binario por cada ingrediente.
Un mutex no sería útil, ya que solo asegura la mutua exclusión, se inicializa SIEMPRE en estado desbloqueado, a diferencia del binario el cual se le puede indicar al crearlo cuantas instancias iniciales de ese recurso hay. Y un semáforo contador no tiene lugar ya que solo va a tener solo una instancia de cada ingrediente y es importante el orden de los mismos.
Es importante que noten que los semáforos SIEMPRE están cruzados, uno hace post del paso siguiente, el cual establoqueadoo en un wait esperando a que sea habilitado por otro hilo.

Ejercicio 8

El error en la solución original radica en la forma de serializar la estructura. Este error existe SOLO porque la estructura ***t_spock*** tiene campos dinámico (punteros) como campos.

Cuando uno hace ***fwrite(file, size, 1, data)*** el ***fwrite*** agarra el comienzo del bloque apuntado por data y escribe los siguientes n bytes.
Si la estructura solo tuviera campos estáticos (como es el caso de la estructura ***t_villano***) puede ser enviada directamente ya que los campos estáticos de las estructuras se acomodan todos correlativos, y por ende pueden ser escritos correctamente por el ***fwrite***.
En el caso de la estructura ***t_spock*** la cual tiene campos dinámico, el ***fwrite(..)*** se comporta de la misma manera, pero el problema que los valores de los campos son los punteros a los datos (por ej: en el campo spock->nombre no se encuentra la cadena “Roberto Spock”, sino que se encuentra un puntero a la cadena). Si nosotros hacemos un ***fwrite*** directamente como hace la solución lo que enviamos es la dirección de memoria del puntero y no el contenido de la estructura asociada. Las direcciones tienen sentido en el contexto de ejecución del proceso, por lo que si le envío a otro proceso una dirección, este no podrá hacer nada con ella, ya que no pertenece al espacio de direcciones original.

Es importante importante ver como en la solución propuesta se usa la función ***list_iterate*** para no romper el encapsulamiento de la lista.