Universidad de San Andrés

I301 - Arquitectura de computadoras y Sistemas Operativos

TP2 - parte 2

Fecha de entrega: **28/09/2025**

Link al código:

https://github.com/nicoRomeroCuruchet/TP2_Parte2/tree/main

Importante: les pedimos que mantengan los grupos en el TP1. La entrega será por campus.

Enunciado

El objetivo del trabajo práctico es realizar la implementación de distintas funciones sobre strings en lenguaje ensamblador. El lenguaje ensamblador, al ser de mucho más bajo nivel que los lenguajes más utilizados, como Python y Javascript, nos permite una ejecución de varios órdenes de magnitud más rápida que la de estos lenguajes. En esta instancia del trabajo, deberán implementar las siguientes instrucciones sobre cadenas de caracteres, las cuales están originalmente implementadas en la librería estándar de C :

int32_t strCmp(char* a, char* b): Compara dos strings en orden lexicográfico[1]. Debe retornar 0 si son iguales, 1 si a < b y -1 si a > b.

char* strClone(char* a): Genera una copia del string pasado por parámetro. El puntero pasado siempre es válido, aunque podría corresponderse a la cadena vacía.

void strDelete(char* a): Borra el string pasado por parámetro. Esta función debe tener un comportamiento similar a la función free.

uint32_t strLen(char* a): Retorna la cantidad de caracteres distintos de cero del \emph{string} pasado por parámetro.

struct lista strList(char strings):** Retorna una lista enlazada cuyos nodos contienen un puntero a los strings provistos en el orden original. Es decir, el primer nodo tiene un puntero que apunta a una copia del string en la posición 0 del array, y así sucesivamente. Cada nodo debe contener la longitud (cantidad de caracteres) del string en cuestión. del string correspondiente y un puntero al próximo nodo. El último nodo debe apuntar a NULL. El nodo cabecera de la lista debe apuntar al primer nodo y contener la longitud total de la lista.

Por ejemplo: strList(["hola", "acso")] deberá devolver un *struct lista,* la cual deberá apuntar a un nodo cuyo atributo *string* será un puntero que apuntará a una dirección de memoria que contenga el string "hola", y el atributo *próximo* apuntará a un nodo similar, pero que en su atributo string habrá un puntero que apuntará a una dirección que contenga "acso"

Los headers de las instrucciones mencionadas se encuentran en el archivo checkpoint.h, y el archivo checkpoint.asm contiene el archivo en lenguaje ensamblador listo para implementar las funciones.

[1] método para ordenar secuencias de elementos basándose en el orden alfabético de sus componentes, de forma similar a como se ordenan las palabras en un diccionario

Criterio de aprobación

Para aprobar esta instancia del trabajo, su implementación deberá pasar todos los tests que se encuentran en el archivo tester.c. Allí, se encuentran tests para las instrucciones que deben escribirse en assembly, y no está permitido modificarlos. Las instrucciones implementadas no pueden tener errores de memoria y deben tener la pila alineada antes de cualquier llamada a funciones de C (como malloc). Nos reservamos un posible llamado a coloquio para discutir la implementación del trabajo y terminar de definir la nota de este.

Cómo correr los tests

Al correr el comando *make* en la terminal, se correrán los comandos presentes en el archivo *Makefile*, que se encargará de realizar todos los pasos de preprocesamiento, compilación y linkeo. Luego, parados en el directorio adecuado, con el comando *./runTester* corren los tests correspondientes.
Por otro lado, el comando *make clean* elimina los ejecutables y los archivos .o.

Observaciones

Van a encontrar en el directorio del proyecto los archivos checkpoint_c.h y checkpoint_c.c. Allí están definidos los headers de las funciones a implementar y el archivo en el que pueden implementar las funciones en C en caso de así desearlo. Sin embargo, los tests sólo tendrán en cuenta las implementaciones en asm. Pueden usar las implementaciones en C como guía previo a escribir el código en assembly.