Programación Orientada a Datos - ANSI C

Organización del Computador II

Segundo Cuatrimestre 2021

En este taller vamos a trabajar con código `C` interpretándolo desde la perspectiva de los datos, y en particular, de la forma en que los datos se ubican en memoria. Algunos ejercicios deben completarlos en el código que descargaron para el taller y otros les van a pedir que den una explicación a algún tema específico, en ambos casos intenten expresar con el mayor nivel de detaller y claridad sus respuestas.

Al corregir cada **checkpoint** todos los miembros del grupo deben estar presentes, salvo aquellas instancias en que puedan justificar su ausencia previamente. Si al finalizar la práctica de la materia algún miembro cuenta con mas de un 20 % de ausencia no justificada en la totalidad de los checkpoints se considerará que cursada como desaprobada.

Cada checkpoint se presenta a lx docente asignadx en cada breakout room durante el transcurso de la clase práctica actual o la siguiente, no hace falta entregar los archivos.

1. Inicialización de arreglos

En este checkpoint vamos a estudiar e implementar porciones de código vinculado a la inicialización de arreglos, ya sea de forma dinámica o estática. Pueden encontrar el código vinculado a este checkpoint en `checkpoint1.c`, la declaración de las funciones se encuentra en `checkpoints.h`. La ubicación del código que debe completarse se encuentra indicada con la frase `COMPLETAR:`, recuerden retirar los caracteres de comentario simple `//` o múltiple `/* */`.

a) Al comienzo de la función `checkpoint1()` inicialice estáticamente un arreglo de floats de tamaño `FLOAT_ARR_SIZE`.
```
? float_not_initialized_array[?]
```

b) Complete la llamada a `printf` con los modificadores necesarios para el tipo de dato entero y flotante.
```
printf("\t[%?]:\t%?\n", i, float_static_array[i])
```

c) Inicialice dinámicamente un arreglo de floats de tamaño `FLOAT_ARR_SIZE` haciendo uso de `malloc` y `sizeof`.
```
? ?float_dynamic_array = ?;
```

d) Complete la llamada a `printf` con los parámetros necesarios.
```
printf("\t[%?]:\t%?\n", ?, ?);
```

e) Explique la diferencia entre ambas inicializaciones, `float_not_initialized_array` y `float_dynamic_array`.

Checkpoint 1

2. Introducción al manejo de punteros

En este checkpoint vamos a estudiar e implementar porciones de código vinculado al trabajo con punteros. Pueden encontrar el código vinculado a este checkpoint en `checkpoint2.c`, la declaración de las funciones se encuentra en `checkpoints.h`.

a) Expliquen el funcionamiento de la función `sum_product_array(int32_t *arr, int32_t len, int32_t *ptr_sum, int32_t *ptr_prod)`.

b) ¿Cómo conseguirián reproducir la funcionalidad de `sum_product_array` sin utilizar punteros?

c) Expliquen el funcionamiento de la función `void set_string_at_location(char** destination, char* source)`.

d) Implementen la función `void set_int_at_location(int32_t** destination, int32_t* source, uint32_t length)` que copia el contenido de `source` en la posición de memoria de `destination` al igual que `set_string_at_location` lo hace con strings.

e) Expliquen por qué es necesario pasar el largo del arreglo a `set_int_at_location` pero no a `set_string_at_location`. ¿Se puede resolver pasando pasando un puntero simple (`uint_32 *destination`) en lugar de puntero a puntero?

f) Luego de inicializar y utilizar la estructura dinámica `dynamic_item` debemos liberar la memoria reservada a través de `malloc`. ¿Por qué motivo no es suficiente hacer sólo `free(dynamic_item)`?

g) Describa cómo sería el esquema general de liberación de memoria de una estructura compuesta (`struct` o `array`)?

Checkpoint 2