Programación Orientada a Datos - ANSI C

Organización del Computador II

Segundo Cuatrimestre 2022

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. También vamos a ejercitar el uso de una pila implementada en C. Los ejercicios correspondientes al primer checkpoint deben completarlos a partir del análisis del código que descargaron para el taller, mientras que el checkpoint 2 se concentra en la práctica de programación donde deberán completar el código para obtener un resultado correcto de la aplicación. Finalmente el checkpoint 3 trabaja sobre el debugging.

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 durante el transcurso de la clase práctica actual o la siguiente, al igual que en la primera actividad, se recomienda subir los archivos completos al git (en esta actividad pueden armar un repo para todo el grupo).

Introducción

En el campus encontraran el archivo `ejC2-bundle.v0.1.tar.gz` conteniendo el presente enunciado junto con el código fuente (incompleto) de un sencillo programa que busca imprimir por consola el contenido de diferentes estructuras.

El mismo implementa una pila global que utilizaremos para introducir y trabajar algunos conceptos que nos resultarán de utilidad a medida que avance el cuatrimestre. El código fuente de la pila está disponible en `stack.h` y `stack.c` dónde se implementan algunas funciones útiles que deberemos completar.

Los archivos `student.h` y `student.c` definen las estructuras en cuestión y contienen dos funciones `printStudent()` y `printStudentp()` que también deberemos implementar.

1. Análisis

En este checkpoint vamos a estudiar porciones de código vinculado a la inicialización de arreglos, variables, structs, etc. Ya sea de forma dinámica o estática. También vamos a conceptualizar el funcionamiento de la pila implementada.

1.1. Memoria

a) ¿En qué segmento de memoria (heap, stack, text, etc.) se encuentra cada variable enumerada a continuación? (Las mismas se encuentran en `main.c`)

Variable	Segmento	Justificación
`stack`		
`*stack`		
`st1p`		
Lo apuntado por `stack->pop`		

b) Dibujar un diagrama de la organización en memoria de las estructuras `student_t` y `studentp_t` con la alineación de datos correspondiente (ver `student.h`).

c) ¿Que porciones de memoria deberán ser explícitamente liberadas?

1.2. Pila

Mirando los archivos stack.h

a) ¿Qué variable apunta al segmento de memoria utilizado por la pila para almacenar los datos?

b) ¿Qué ancho tiene la pila (i.e. tamaño del dato que se pushea/popea)?

c) Dibujar un esquema de la pila, con su organización en memoria y sus miembros.

Checkpoint 1

2. Práctica de programación

En este checkpoint vamos a implementar porciones de código vinculado al trabajo con datos y uso de una pila.
Se pide: Completar `main.c`, `student.c` y `stack.c` para que:

a) El programa pueda imprimir correctamente los estudiantes y el profesor pusheados, por ejemplo:

```
Nombre del profesor: Alejandro Furfaro
Nombre: Linus Torvalds
dni: 23456789
Calificaciones: 9, 7, 8,
Concepto: 1
----------
Nombre: Steve Balmer
dni: 12345678
Calificaciones: 3, 2, 1,
Concepto: -2
----------
```

Checkpoint 2

3. Debugging

a) Utilizando `gdb`, insertar un breakpoint en la primera línea de la función `push` del stack, usando el comando `break` (consultar su utilización con `help break`). Correr el programa (`run`).

b) En el breakpoint insertado en el punto anterior, ejecutar la siguiente línea en `gdb`: `print *((student_t*) $rsi)` Discutir el resultado de dicho comando. ¿Por qué se imprime eso?

c) Sin avanzar con la ejecución del programa, imprimir el contenido de la estructura `stack` y explicar el resultado. Recordar usar `help print` si es necesario

d) Verificar el correcto manejo de la memoria con la herramienta Valgrind. La línea para ejecutarlo es:

```
valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes --verbose ./app
```

Checkpoint 3