Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Actividad 0

IIC2133 - Estructuras de datos y algoritmos

Primer semestre, 2018

Entrega: Domingo 18 de marzo

Introducción

¡Bienvenido a la primera actividad! A modo de familiarizarse con el lenguaje de programación C, en esta actividad deberás implementar las funciones básicas de dos tipos distintos de **listas**: una *array list* y una lista ligada. Se espera que logres manejar las estructuras usando punteros y que aprendas a manejar la memoria de su programa de manera responsable y eficiente. También tendrás que analizar el tiempo de ejecución de tus estructuras al realizar las distintas operaciones y hacer un informe con tus conclusiones.

La actividad se dividirá en 2 ayudantías distintas por lo que se pretende que implementes una estructura en cada sesión. Tus dudas las puedes preguntar en la ayudantía o a través del foro de github (No se responderán dudas a última hora por ningún medio).

Array List - Viernes 9

Una *Array List* un tipo de lista que almacena todos sus datos en un `arreglo`. Al llenarse el arreglo, se aumenta su tamaño (usualmente al doble) para que pueda almacenar más datos y se traspasan todos sus elementos al arreglo nuevo. Recuerda que un arreglo tiene todos sus datos adyacentes en memoria.

Lista Ligada - Viernes 16

Una lista ligada almacena los datos en nodos, los cuales tienen referencias al elemento siguiente. Esto permite a la estructura crecer un nodo cada vez que se inserta algo, sin necesitar haya memoria adyacente disponible.

Operaciones

Para ambas versiones de las listas deberás implementar las siguientes operaciones:

- `init()`: Crea una lista vacía.
- `append(list, element)`: Insertar un elemento al final de la lista.
- `get(list, i)`: Obtiene el elemento almacenado en la posición `i`.
- `insert(list, element, i)`: Insertar elemento en posición `i`.
- `delete(list, i)`: Elimina elemento en la posición `i`.
- `concatenate(list1, list2)`: Concatena a lista `list1` la lista `list2`.
- `destroy(list)`: Libera **todos** los recursos asociados a una lista.

Memoria

Se espera que su programa maneje correctamente la memoria, esto quiere decir que no puede tener `leaks`. Para ayudarte con esto, puedes usar la herramienta `valgrind`, que analiza los errores y el uso de memoria de tu programa.

Archivos

Se te entregarán 3 clases de archivos:

- `arraylist.h y linkedlist.h`: En los archivos `.h` se definen estructuras y los `headers` de sus funciones. En estos archivos solo deben completar la estructura correspondiente (struct arraylist o struct linkedlist).
- `arraylist.c y linkedlist.c`: En los archivos `.c` se completan las funciones definidas por los headers de los archivos `.h` y se pueden definir funciones extra que sirvan para el comportamiento interno de la estructura.
- `main.c`: Este archivo importa a `arraylist.h y linkedlist.h`, por lo que se pueden llamar todas las funciones definidas ahí. En este archivo además se define la función `main`, que recibe el input del programa y retorna 0 si todo esta bien y otra cosa si hubo un error.

Adicionalmente se les entregará un archivo llamado `Makefile` que sirve para compilar fácilmente el código. Este archivo no debe ser editado y no es necesario que lo entiendas. Para poder usar tu programa primero deberás compilarlo desde Terminal, para esto utiliza el siguiente comando en el directorio respectivo: `make`. Luego, puedes ejecutar el programa compilado llamándolo con `./tu_programa`. Cualquier variable que agregues a la derecha del nombre del programa irá a los argumentos de tu función `main`. Puedes agregar `time` a la izquierda para obtener el tiempo de ejecución del programa, por ejemplo:

```
time ./tu_programa
```

Ejecutar tu programa con `valgrind` sigue el mismo formato:

```
valgrind ./tu_programa
```

Informe

En tu informe deberás analizar el tiempo que toma cada operación en ambas estructuras y compararlos. Para esto puedes usar la librería `time.h` de C o simplemente ejecutar tu código con el comando `time`. Luego de tomar los tiempos debes decir si los tiempos se ajustan a las complejidades esperadas de las operaciones y por qué, incluyendo **gráficos** pertinentes. Se recomienda que pruebes tu programa con distintas cantidades de números para poder analizarlo mejor. Considera también probar los peores casos de cada operación para cada estructura.

Entrega

Se te asignará un repositorio en github para la actividad en el cual deberás subir tu código y tu informe hasta el domingo 18 de marzo a las 23:59. No se aceptarán entregas atrasadas. Recuerda contestar la encuesta del SIDING con tu usuario de github para que tengas acceso a tu repositorio.

Evaluación

La nota de la actividad se calcula de la siguiente manera:

- `Informe` (50 %):
  - Análisis empírico de las operaciones en la array list (1 punto cada una).
  - Análisis empírico de las operaciones en la lista ligada (1 punto cada una).
  - Análisis de los resultados según sus complejidades esperadas (10 puntos).
  - Comparación de las estructuras y conclusiones (10 puntos).
- `Código` (50 %):
  - Array list: 1 punto por cada operacion.
  - Lista ligada: 1 punto por cada operacion.
  - Uso de memoria: 10 puntos por no tener leaks ni errores de memoria.