Manual Ejercicios Algorítmica y Estructura de Datos
Entrega 1

Elias Blanco Castro
Simón Suárez Rosende

Ejercicio 1 y Ejercicio 2:

Estos dos archivos se desarrollan en el mismo archivo el cual es la clase de puertas, estan desarrollados en python y básicamente lo que se hizo fue incluir más puertas creando una clase específica para cada una y heredando de la clase puerta binaria. Las lógicas de puerta serían básicamente para la NAND que si ambas entradas son 1 la salida es 0 (si no es 1), para la NOR que si alguna de las entradas es 1 la salida es 0(si no es 1) y por último para la xor que si ambas entradas suman 1 la salida es 1 (si no es 0). Todo esto para conseguir las salidas que nos ofrecen estas puertas en la realidad. A parte de esto se realizó una comprobación de datos para que, en caso de que el número sea distinto de 1 o 0, lance una excepción que informe al programa que tira de la clase. El ejercicio dos nos pedía comparar dos funciones lógicas y ver si el resultado es el mismo, para ello se modificó la clase de forma que los datos los introdujese directamente el programa y no el usuario para así poder comparar todas las 16 combinaciones posibles. Las distintas combinaciones las hacemos mediante bucles for, los resultados se guardan en dos listas distintas y al final se compara si son iguales para dar el resultado final.

Ejercicio 3 y Ejercicio 4:

Estos ejercicios los hicimos en el mismo archivo pues ambos consistían en editar la clase fracción. Presentamos dos programas: “ej3y4_clase” y “ej3y4_programa”. Introducimos los métodos simples getNum, getDen y los operadores relacionales: (>, >=, <, <=, y !=) a través de sus métodos mágicos y en el “ej3y4_programa” mostramos cómo funcionan. Para utilizar el código solo hay que introducir los datos en las variables x e y como una lista, en la que el primer elemento será el numerador y el segundo el denominador. Después se aplican todos los métodos simples sobre ambas fracciones. funcionando los últimos como booleanos (True or False).
Después modificamos la clase para que el número introducido en el denominador pudiese ser negativo, como muestra el ejemplo por defecto que presentamos.

Ejercicio 5:

Este código tiene como referencia una fecha que utiliza como ejemplo de fecha correcta, luego solicita el día, el mes y el año, y si los datos introducidos no son correctos muestra que la fecha es incorrecta.

Ejercicio 6:

En el ejercicio 6 nos piden implementar una clase para las horas, en esta lo que hacemos es implementar la introducción de horas, minutos y segundos y crear excepciones para que sea robusta (que las horas no superen de 23, no sean menores de 0, los segundos no sean mayores de 60…).
Aparte, también se ha implementado un formato de impresión en pantalla, una función que comprueba en qué parte del día estamos (mañana, tarde, noche y madrugada) y una función que compara dos horas introducidas y te indica si la primera es menor, mayor o igual que otra introducida. En el programa de prueba esa hora que comparamos es la actual que obtenemos mediante la librería datetime y la función datetime.now().

Ejercicio 7

En el ejercicio 7 desarrollamos la clase polinomio en C++, a la cual introduciremos los datos en dos vectores(con el formato x^3+x^2+x+término independiente), y entonces el programa primero comprobará si los vectores son iguales, después los sumará y por último los restará.

Ejercicio 8

Esta clase recibe 2 parámetros los cuales serán un número real y otro complejo, si solo recibe uno real, se asimila que el complejo es 0 y si no recibe ningún parámetro se asimila que directamente ambos son 0. Desarrollamos una función de devuelve el conjugado, esta simplemente cambia de signo el número complejo. A parte mediante “operator” podemos asimilar las operaciones suma, resta, multiplicación y división y crear funciones que realicen cada una de ellas.
-La suma suma los dos reales y los dos complejos.
-La resta resta los dos reales y los dos complejos.
-La multiplicación multiplica los dos reales y se lo suma a la multiplicación de los dos complejos (se pueden sumar ya que ahora son reales) y multiplica los reales con los complejos de forma cruzada(el resultado de esta operación sería la parte imaginaria).
-La división crea un denominador real y multiplicando por el conjugado tanto el primer como el segundo complejo obtenemos el resultado.
Para imprimir el complejo en pantalla lo hacemos mediante el ostream y condicionamos para que si el complejo es negativo no se imprima el signo más en pantalla.
En cuanto al programa que tira de la clase lo único que hace es imprimir en pantalla los resultados de las operaciones anteriores y hace el conjugado.

Ejercicio 9

En el ejercicio 9 creamos varías clases para llevar a cabo el juego,en primer lugar la clase dado que genera un número aleatorio entre 1 y el número de caras del dado, tomamos también como el número de dados que se tiran. Después creamos la clase jugador que representa a un jugador que tiene x datos, y permite introducir todos los jugadores que deseemos ,finalmente aplicamos ambas clases en la clase Juego, que se inciará con unos nombres de ejemplo y nos pedirá que introduzcamos n(número de caras y de dados) y x número de turnos que se jugarán.

Ejercicio 10

Aquí lo que hacemos es similar a los ejercicios 3 y 4 pero en c++. Aquí aprovechamos la clase en c++ ya hecha de fracciones en c++ y mediante operator establecemos las operaciones básica +, -,* y / y mediante ellos creamos funciones que sirve para realizar estas, a parte introducimos los distintos comparadores (==, !=, <, <=, >, >=) que también mediante funciones realizarán las distintas comparaciones entre estos. A parte creamos una función getNum y otra getDem, para hacerlo y que no nos devuelva una fracción, en el ostream creamos una condición para que si el denominador es igual a 1 solo nos devuelva el numerador.

Ejercicio 11:

Este programa utiliza dos métodos para comparar los elementos de una lista, el primero tiene eficiencia O(n²) ya que utiliza dos bucles for y va comparando cada elemento de la lista con los demás, el segundo tiene eficiencia O(n) ya que para ello simplemente utiliza un bucle for.

Ejercicio 12:

Este código aumenta progresivamente el tamaño de una lista y mediante la función timeit.Timer mide el tiempo que tarda la indexación de un número aleatorio dependiendo del tamaño de la lista. La conclusión es que la indexación tiene eficiencia O(1) ya que el tiempo siempre es el mismo independientemente de la lista.

Ejercicio 13:

En el ejercicio 13 tratamos de diseñar un experimento para verificar que las operaciones de obtención y asignación de ítems para diccionarios son $\boldsymbol{O(1)}$ para ello creamos una función que va tomando los tiempos(mediante la librería time) que tardan en obtenerse y asignarse los elementos y crea una tabla, cuyas columnas son, la dimensión del item, la mejor obtención, la mejor asignación, el tiempo de obtención medio, el tiempo asignación medio, el peor de obtención y por último el tiempo de peor asignación.

Ejercicio 14:

En este ejercicio utilizamos de nuevo timeit.Timer pero en esta ocasión para medir el tiempo que lleva a las listas y diccionarios eliminar un elemento. En principio se tenían que realizar ambos mediante del, no obstante al utilizar del en el diccionario ocurría una excepción extraña por lo que al final nos decantamos por utilizar pop. En este programa se ve claramente como la eliminación en un diccionario es de orden O(1) mientras que en una lista es de orden O(n).

Ejercicio 15

En este ejercicio desarrollamos un programa que dada una lista de números en orden aleatorio encuentra el k-ésimo número más pequeño de la lista, mediante un bucle for y a partir del elemento k busca en la lista cuál es el k-ésimo elemento más pequeño en el orden establecido.

Ejercicio 16

No fuimos capaces de realizar el ejercicio anterior para $\boldsymbol{O}$(nlog(n)), tendríamos que haber utilizado un while en la función principal.

Ejercicio 17:

Primero creamos la función que suma los componentes del vector y después los títulos de la tabla, a continuación el programa calcula la media de varios de los tiempos de ejecución en la función, por último devuelve la suma de los elementos del vector y mostramos la notación asintótica.

Ejercicio 18:

Por último, este ejercicio consiste en crear una función que busque los valores similares y el número de veces que se repiten en una lista y en un diccionario, calculamos el tiempo de ambas mediante la librería time, mirando la diferencia entre el tiempo final y el tiempo inicial. La conclusión es que la forma que tiene el tiempo de subir es parecida a lo que sería la eficiencia O(n logn), a parte vemos que los diccionarios son mucho más rápidos en ejecución que las listas.