UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructuras de Datos I

Taller de Vectores - Entrada/Salida

1. Indicaciones

El objetivo del taller es resolver ejercicios en los cuáles tendrán que trabajar con vectores, entrada y salida estándar y manipulación de archivos. Podrán encontrar los materiales para el taller de hoy en el archivo (`template-alumnos.zip`), en el folder correspondiente a este laboratorio. Una vez descomprimido el archivo, se encontrarán con el siguiente árbol de directorios:

- `template-alumnos/archivos`
    1. `archivos/`: carpeta con archivos con datos, secuencias de números y palabras separados por espacios.
- `template-alumnos/src`
    1. `src/main.cpp`: El archivo principal, utiliza los módulos `#include "generador.h"` y `#include "vectores.h"` y .
    2. `src/generador.cpp`: Contiene una función que se puede llamar desde el main para generar automáticamente en la plataforma online los archivos numéricos para los ejercicois.
    3. `src/generador.h`: Contiene la declaración de la función de generación de archivos.
    4. `src/vectores.cpp`: Contiene el esqueleto de algunas de las funciones a implementar.
    5. `src/vectores.h`: Contiene la declaración (o headers) de los funciones a implementar en `vectores.cpp`. Pueden agregar más funciones si lo necesitan.

1.1. Plataforma online

Para la utilización de este codigo en la plataforma online:

1. Copiar y pegar el codigo del archivo `main.cpp` dado por la cátedra sobre el `main.cpp` de la plataforma.
2. Utilizando el icono de "New File" o Ctrl+M, crear los archivos `generador.cpp`, `generador.h`, `vectores.cpp` y `vectores.h`, y copiar alli el código dado por la cátedra para cada uno de ellos.
3. Una vez creados los archivos base, utilizando el icono "Save" o Ctrl+S, se graba el Proyecto, por ejemplo, con el nombre "labo_vectores". De esta manera pueden volver a leerlo luego desde el panel del menú, a la izquierda, y no precisan copiar todos los archivos nuevamente. Conviene generar un usuario para ello.
4. En el archivo `vectores.cpp` se pueden empezar a implementar los ejercicios que se llamarán desde el main, o se pueden llamar también desde la función "holaModuloVectores" que se ejecuta desde el main.
5. Cuando se llega a la sección de ejercicios de entrada-salida, se precisa el uso de los archivos auxiliares. En este punto se puede descomentar en el archivo `main.cpp` la línea de generación automática para los archivos numéricos que ejecuta la función `generarDatosLaboratorio()`. En caso contrario se debe crear con "New File" el archivo y compiarle el contenido. Para el archivo de texto `cantidadAparicionesDePalabra.in` esto se debe hacer a mano. Una vez creados los archivos se pude volver a comentar la línea de generación, para que no se ejecute cada vez que se prueba una función.
6. Se recomienda crear un nuevo archivo, .cpp y .h, para la sección de entrada y salida.
7. Desde el panel del menu, a la izquierda de la plataforma online, cuando se hace click sobre "My Projects", se puede acceder a los proyectos que tenemos guardados, donde se pueden borrar, descargar o volver a poner sobre la plataforma.

1.2. CLION

Aquellos que estén utilizando CLion van a tener que hacer lo siguiente para poder utilizar el código de este labo.

1. Descomprimir el .zip. Por ejemplo les queda: [ruta]/template-alumnos/src y [ruta]/template-alumnos/archivos
2. Lanzar el CLion y en el menu de inicio deben seleccionar Open.
3. En la ventana de Open File or Project seleccionar '[ruta]/template-alumnos'y luego OK.

4. Se abre el CLion. Abrir src/main.cpp y aparecerán las opciones 'Select CMakeLists.txt'y 'Create CMakeLists.txt'. Hacer click en 'Create CMakeLists.txt'y OK.

5. Se crea automáticamente el archivo CMakeLists.txt con los archivos .cpp y .h del laboratorio. Algo así

Archivo: CMakeList.txt

```
cmake_minimum_required(VERSION 3.14)
project(template-alumnos)

set(CMAKE_CXX_STANDARD 14)

include_directories(.)

add_executable(template-alumnos
        generador.cpp
        generador.h
        main.cpp
        vectores.cpp
        vectores.h)
```

Para que el compilador guarde el ejecutable en la carpeta `template-alumnos` y funcionen las rutas relativas (`archivos/...`), agregar en el archivo `CMakeList.txt` **solo una** de las siguientes opciones:

1. `set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR})`
2. `set(EXECUTABLE_OUTPUT_PATH ${CMAKE_CURRENT_SOURCE_DIR})`

Luego hacer click en 'Reload changes'. Verificar que al compilar, el ejecutable quede guardado en el directorio raiz.

2. Ejercicios

Aclaraciones:

- Salvo en el caso que lo pida explícitamente, no "convertir" los datos del archivo de entrada en un vector.
- Para todos los ejercicios que requieran utilizar la entrada y/o salida estándar escribir el código necesario en main.cpp para realizar las acciones pedidas.
- En esta práctica utilizamos la clase `string` que va a contener una cadena o vector de caracteres (tipo `char`). Para poder utilizarla, hay que incluir en el encabezado `#include <string>`. Podemos definir una variable de tipo string como `string mi_variable;` y luego asignarle un valor `mi_variable = "archivo.txt"`, o incluso, esta clase puede recibir toda una cadena de caracteres por teclado: `cin >> mi_variable;`

2.1. Vectores

1. `bool divide(vector<int> v, int a);`

   Dados un vector `v` y un entero `a`, decide si `a` divide a todos los elementos de `v`.

2. `int mayor(vector<int> v);`

   Dado un vector `v`, devuelve el máximo.

3. `vector<int> reverso(vector<int> v);`

   Dado un vector `v`, devuelve el vector reverso.

4. `vector<int> rotar(vector<int> v, int k);`

   Dado un vector `v` y un entero `k`, rotar `k` posiciones los elementos de `v`. Ejemplo: `<1,2,3,4,5,6>` rotado 2, debería devolver `<3,4,5,6,1,2>`.

5. `bool estaOrdenado(vector<int> v);`

   Dado un vector `v` de enteros no repetidos, devuelve verdadero si el mismo está ordenado (ya sea creciente o decrecientemente).

6. `vector<int>factoresPrimos(int n);`

   Dado un entero que debe ser leído de la entrada estándar, escribir la función factoresPrimos que devuelve un vector con los factores primos del entero. Mostrar el resultado por pantalla.

7. `void mostrarVector(vector<int> v);`

   Dado un vector de enteros `v` muestra por pantalla su contenido. Ejemplo: si el vector es `<1, 2, 5, 65>` se debe mostrar por pantalla `[1, 2, 5, 65]`.

2.2. Lectura y escritura

8. `vector<int> leerVector(string nombreArchivo);`

   Dado un archivo que contiene una secuencia de números enteros separados por espacio (por ejemplo: `1 2 34 4 45`), leerlo y devolver un vector con los números en el mismo orden.

9. `void guardarVector(vector<int> v, string nombreArchivo);`

   Dado un vector de enteros, grabar sus elementos en un archivo cuyo nombre viene como parámetro de entrada. Ejemplo: si el vector es `<1, 2, 5, 65>` el archivo contiene `[1, 2, 5, 65]`.

10. `int elementoMedio(vector<int>v);`

    Dado un vector de enteros encontrar el primer elemento de izquierda a derecha tal que los elementos a su izquierda suman más que los que están a su derecha. Ejemplo: <1, 2, 3, 4> el resultado es 3 porque (1+2) < 3 + 4 y (1 + 2 + 3) > 4. El vector de entrada debe ser leído desde un archivo y el resultado debe ser mostrado por pantalla. Utilizar el archivo `elementoMedio.in` para probar la función.

11. `void cantApariciones(string nombreArchivo);`

    Dado un archivo que contiene una lista de números, contar la cantidad de apariciones de cada uno y crear en un archivo en el directorio `archivos/out` con el mismo nombre del archivo de entrada, de manera de tener una línea por cada número encontrado, un espacio y su cantidad de apariciones.

    Por ejemplo, si el vector es `<1, 2, 2, 1, 1, 4>` el archivo de salida tiene que ser:

    ```
    linea 1: 1 3
    linea 2: 2 2
    linea 3: 4 1
    ```

    Utilizar los archivos `10000NumerosEntre1y50.in` y `cantidadApariciones.in`.

12. `int cantidadAparicionesDePalabra(string nombreArchivo, string palabra);`

    Ingresar por consola una palabra a buscar y el nombre de un archivo de texto y devolver la cantidad de apariciones de la palabra en el archivo. Mostrar el resultado por pantalla.
    Para testear el ejercicio pueden usar el archivo `cantidadAparicionesDePalabra.in`.

13. `void promedio(string nombreArchivoIn1, string nombreArchivoIn2, string nombreArchivoOut);`

    Dados dos archivos en los que cada uno contiene una secuencia de enteros de la misma longitud, guardar el promedio de cada par de números que se encuentran en la misma "posición" en el archivo de salida. Ejemplo: si tenemos dos secuencias `<1, 2, 3, 4>` y `<1, 25, 3, 12>` el resultado debe ser [1, 13.5, 3, 8]. En `archivos/` se encuentra `promedio1.in` y `promedio2.in`. Cada archivo contiene 100 números random entre 1 y 10.

14. `void ordenarSecuencias(string nombreArchivoIn1, string nombreArchivoIn2, string nombreArchivoOut);`

    Dados dos archivos en los que cada uno contiene una secuencia de enteros ordenada, ordenarlos y guardar el resultado en el archivo de salida. Ejemplo: si tenemos dos secuencias `<1, 4, 8, 19>` y `<3, 25, 31>` el resultado debe ser `[1, 3, 4, 8, 25, 31]`.

    En `archivos/` se encuentra `ordenarSecuencia1.in` y `ordenarSecuencia2.in`. Cada archivo contiene 5000 números ordenados entre 1 y 1000. El primer archivo contiene los números pares en el rango y el segundo los impares.

15. `vector<int> interseccion();`

    Función que pide al usuario que se ingrese por teclado dos nombres de archivos que contengan solo números enteros, luego calcule la intersección (los elementos comunes a ambos archivos), que debe mostrar por pantalla, además de devolverla como vector.