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

Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2019

9 de septiembre de 2019

Taller de Entrada/Salida - Vectores

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 (`labo04_src.zip`), en la sección descargas de la materia. Una vez descomprimido el archivo, se encontrarán con el siguiente árbol de directorios:

- `template-alumnos/src`
  1. `src/vectores.cpp`: Contiene el esqueleto de algunas de las funciones a implementar.
  2. `src/vectores.h`: Contiene la declaración (o headers) de los funciones a implementar en `vectores.cpp`. Pueden agregar más funciones si lo necesitan.
  3. `src/main.cpp`: El archivo principal, utiliza el módulo `#include "vectores.h"`.
- `template-alumnos/archivos`
  1. `archivos/in/`: carpeta con archivos con datos, secuencias de números separados por espacios.
  2. `archivos/out/`: carpeta donde guardar los archivos de salida de sus funciones.

Crear un nuevo proyecto en `CLion`, eligiendo como destino el directorio `template-alumnos`, y agregar al mismo `vectores.cpp` y `vectores.h`.

> Para que el compilador guarde el ejecutable en la carpeta `template-alumnos` y funciones las rutas relativas (`archivos/in/...`), 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})`
>
> Verificar que al compilar, el ejecutable quede guardado en el directorio raiz.

2. Ejercicios

Aclaraciónes:

- 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.

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, devuelve verdadero si el mismo está ordenado (ya sea creciente o decrecientemente).

6. `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

1. `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.

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

   Dado un vector de enteros y el nombre de un archivo de salida, escribe al vector en el archivo cuyo nombre se recibe como parámetro. Ejemplo: si el vector es `<1, 2, 5, 65>` el archivo contiene `[1, 2, 5, 65]`.

3. `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`.

4. `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`.

5. `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/in/` se encuentra `promedio1.in` y `promedio2.in`. Cada archivo contiene 100 números random entre 1 y 10.

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

7. `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 y la devuelve como vector.