Documentación del Proyecto N°1 de “Programación en Lenguaje C”

Comisión #17

Alumnos

David Emanuel Latouquette - LU: 118801

Otto Krause - LU: 135758

CONTENIDO

1. Definición y especificación del Proyecto

1.1. Definición general del Proyecto de software

El proyecto de software consiste en el desarrollo de un programa que permita indicar un directorio existente en la computadora y a partir de dicho directorio, leer todos los archivos de textos existentes y poder contabilizar las repeticiones de las palabras de dos maneras distintas:

1. ***Contabilizar repeticiones por agrupadas por archivo.*** Este conteo se realiza sobre las palabras de un mismo archivo y se realiza de manera independiente para cada uno de los archivos de texto del directorio dado.

2. ***Contabilizar el total de las repeticiones entre todos los archivos.*** Este conteo de repeticiones implica contabilizar y sumar las repeticiones de las palabras entre todos los archivos de texto y volcar dicha información en un solo lugar.

La funcionalidad principal del sistema de software se basa en generar dos archivos como resultados de estos dos conteos de repeticiones de palabras, por caso:

1. ***Archivo cadauno.out:*** En este archivo se plasmará el primer caso mencionado, en donde se agrupará por nombre de archivo cada una de las palabras con sus respectivas repeticiones.

2. ***Archivo totales.out:*** En este archivo se plasmará el segundo caso mencionado, en donde todas las palabras son contabilizadas entre todos los archivos.

1.2. Especificación de requerimientos del proyecto

I. Definición general

Idea general. La idea general del presente proyecto de software es realizar la implementación, en el lenguaje de programación C, de un sistema que denominado ***Cuentapalabras*** que permita realizar el conteo de las palabras tal como se ha mencionado en 1.1. Definición general del Proyecto de software.

Objetivos. El objetivo principal del presente sistema se enfoca en desarrollar un programa que respete lo mencionado en 1.1. Definición general del Proyecto de software, y que al mismo tiempo, permita aprovechar la portabilidad del lenguaje C para que el producto resultante pueda ser exportado a cualquier sistema operativo.

> Por tal motivo, la presente documentación tiene la función de brindar al usuario una guía para que pueda conocer mejor al sistema, lo que implica conocer su funcionamiento, sus limitaciones, los requerimientos de instalación y como se utiliza el programa.

Usuarios. Este sistema puede ser utilizado por cualquier persona que tenga conocimiento intermedio del uso de *Intérpretes de Comandos* (CMD en Windows y Terminal en Linux), lo que será importante para poder instalar el programa y utilizarlo correctamente. Estos temas se verán más adelante en 3. Descripción de procesos y servicios.

II. Especificación de requerimientos del proyecto de software

Requerimientos generales. Este proyecto de software ha requerido del uso de distintos TDAs[1] para poder estructurar los datos y así poder almacenar y hacer uso de dichos datos de una manera sencilla.

> Por caso, se necesita de dichas estructuras en el presente sistema para poder contener los elementos, que serán pares *(palabra, cantidad de repeticiones)*, los cuales deben ser ordenados y manejados empleando el uso de memoria de una forma eficiente.

Requerimientos funcionales. El presente sistema de software tiene una sola función y es la de leer los archivos de textos contenidos en un directorio y a partir de dichos archivos, se contabilizan las repeticiones por archivo y en total de cada palabra, para luego plasmar dicha información en dos archivos.

> La única interacción que el usuario tendrá con el sistema es al invocarlo, ya que debe especificar un parámetro y un directorio existente para que el programa luego haga el resto.

Información de autoría y Legacy del proyecto. Este proyecto de software ha sido desarrollado empleando la especificación de las operaciones de cada TDA (y de las estructuras de datos con las que son implementadas) brindadas por la cátedra con lo que se pudo realizar la construcción de los archivos de encabezados de los TDAs y parte de la implementación.

> A su vez, en la implementación se ha debido solucionar problemas haciendo uso de librerías que fueron importadas en el proyecto.

Alcance del sistema. Si bien el lenguaje de programación permite generar programas multiplataformas, la verdad es que solo ha sido testeado bajo los sistemas operativos de Windows, por lo que no se puede asegurar su compatibilidad con otros sistemas operativos.

> Dicho esto, se asegura que hay total compatibilidad del proyecto de software con la plataforma previamente mencionada.

[1] Un **TDA** (Tipo de Dato Abstracto) modela un conjunto de operaciones que dispone un determinado conjunto de datos, pero sin necesidad de revelar cómo es implementada dicha funcionalidad.

III. Especificación de procedimientos

Herramientas utilizadas: En el desarrollo del proyecto de software se ha utilizado como entorno de desarrollo integrado (IDE) el software ***Code::blocks***, el cual permite construir proyectos en el lenguaje de programación C y poder compilarlos y probarlos fácilmente, sin necesidad de tener que hacer el procedimiento de compilación por línea de comandos.

Planificación: A grandes rasgos, la planificación del desarrollo del proyecto consistió en una serie de procesos que se tuvieron que realizar y que serán mencionados a continuación.

1. Construcción de los TDAs[2] Lista y Multiset a partir de la información brindada por la cátedra.
2. A partir de la definición del TDA Lista, realizar su implementación con una lista simplemente enlazada y sin nodo cabecera.
3. A partir de la definición del TDA Multiset, realizar su implementación empleando una estructura de árbol trie de hasta 26 hijos.
4. A partir de las implementaciones ya testeadas de Lista y Multiset, se procede a realizar la implementación del sistema Cuentapalabras.
    a. Lectura de los archivos y creación de los multisets.
    b. Obtención de los datos del multiset en listas.
    c. Creación de los archivos de salida a partir de las listas en cuestión.

Haciendo un poco más de hincapié en la planificación, estos items implican las siguientes tareas a desarrollar por el grupo:

1. Para la construcción del TDA Lista y TDA Multiset se emplearía la información brindada por la cátedra para el proyecto, lo que permite definir dichos TDAs.
2. La implementación del TDA Lista consiste en realizar la implementación de dicho TDA, que en este caso es empleando una lista simplemente enlazada sin nodo centinela y que contendrá elementos y un indicativo de cuál es el nodo siguiente.
3. La implementación del TDA Multiset consiste en realizar la implementación de dicho TDA empleando un árbol ***trie***, el cual cada nodo contendrá hasta 26 hijos y cada uno representará una letra entre la **a** y la **z** (exceptuando la ñ).
4. La implementación de Cuentapalabras consiste en realizar un programa que leerá cada archivo de texto, recopila las palabras y repeticiones en los multisets (uno por cada archivo) y finalmente escribirá la información devuelta de manera ordenada ascendentemente en los archivos de salida mencionados previamente.

[2] TDA es un acrónimo para referirse a Tipo de Dato Abstracto.

2. Arquitectura del Sistema

2.1. Descripción jerárquica

Puesto que la organización de los archivos (tanto fuentes como de encabezados) están todos en el mismo sitio y al construir el programa formar todo una sola pieza de ejecución, se puede concluir que la estructura del sistema es una ***estructura monolítica.***

2.2. Descripción individual de los módulos

I. Módulo TDA Lista

En el presente proyecto, el ***TDA Lista*** es una estructura de datos que tiene la función de almacenar datos de manera secuencial.

> La implementación de este TDA Lista se realiza empleando una lista ***simplemente enlazada y sin nodo cabecera***, lo que implica que todo nodo solamente conoce el nodo siguiente y que hay que tener cuidado de que sucede en los casos de inserción sobre el inicio y final de la lista.

Este TDA Lista hace uso del concepto de ***posición indirecta*** para referirse a la posición de un nodo, esto es, si se quiere acceder a la posición del nodo con posición 2, se recorre secuencialmente hasta el nodo anterior, que sería la posición indirecta del nodo 2 y es posible acceder al nodo siguiente empleando la referencia al siguiente nodo.

> Esta dirección indirecta trae consigo el beneficio de poder realizar asignaciones a nuevos nodos o eliminarlos de manera más directa y sin necesidad de hacer varios recorridos auxiliares.

Además, este TDA modela operaciones cuya finalidad es poder manipular elementos que serán pares *(entero, cadena)* que en el contexto del sistema serán una palabra con la cantidad de repeticiones de dicha palabra.

Función específica. Este TDA Lista tiene como principal funcionalidad almacenar una lista de elementos (como se ha dicho previamente) compuestos de pares *(entero, cadena)*.

Implementación. La implementación de este TDA está realizada en el archivo ***lista.c***, mientras que en el archivo ***lista.h*** está la definición de las operaciones provistas por el TDA.

II. Módulo TDA Multiset

En el presente proyecto, el ***TDA Multiset*** es una estructura de datos que tiene la función de almacenar las cadenas de caracteres leídas desde los archivos y el número de repeticiones de cada cadena.

Este TDA Multiset en su implementación hace uso de una estructura ***trie***, que resulta en un árbol que cada nodo representa un distinto carácter (de la **a** hasta la **z**), por lo que cada nodo tiene a lo sumo 26 nodos.

> Cada nodo tiene un atributo entero que representa la cantidad de repeticiones que tiene la cadena formada por recorrido de la raíz hasta el nodo actual. Si dicho entero es 0, entonces no tiene repeticiones porque no se cargó dicha cadena, en caso contrario, será una cadena con tantas repeticiones.

Toda cadena contenida en el multiset comienza partiendo de la raíz del árbol (que representa una cadena vacía) y va recorriendo los hijos de cada nodo para poder formar palabras y será una palabra válida si la misma tiene al menos una repetición.

Función específica. Este TDA Multiset tiene como principal funcionalidad la de almacenar las cadenas leídas de los archivos empleando un árbol trie que esto permite representar las cadenas cargadas y sus correspondientes repeticiones pero sin la necesidad de mantener almacenada la cadena, sino que la cadena es armada mediante el uso de las posiciones de los nodos.

Implementación. La implementación de este TDA está realizada en el archivo ***multiset.c***, mientras que en el archivo ***multiset.h*** está la definición de las operaciones provistas por el TDA.

III. Módulo Cuentapalabras

En el presente proyecto, el ***Módulo Cuentapalabras*** es la implementación requerida por el proyecto, en donde se realiza el control del directorio dado, así como la recopilación y lectura de los archivos de texto, así como el conteo de las palabras y su posterior escritura en los archivos de salida ***cadauno.out*** y ***totales.out***.

Este módulo hace uso de los TDAs previamente mencionados. En el caso del multiset, este es utilizado con dos fines, almacenar en **n** multiset las repeticiones de cada palabra de cada uno de los **n** archivos de textos encontrados en el directorio, y el otro caso paralelo es tener un multiset que funcione como acumulador para contabilizar el total de repeticiones de cada palabra entre todos los archivos leídos.

Función específica. La función específica de este módulo es la de llevar el hilo de ejecución de cada una de las operaciones que impliquen control del directorio, lectura de archivos, cargar los multiset con las palabras encontradas en los archivos y posteriormente plasmar la información obtenida de los multiset en los archivos en cuestión.

Implementación. La implementación de este TDA está realizada en el archivo ***cuentapalabras.c***, que a su vez dispone de la función **main** para indicar el comienzo de la ejecución desde dicho archivo.

IV. Módulo Define

La única función que tiene este módulo es la de poder definir constantes “booleanas” para las operaciones que requieran un verdadero o falso.

2.3. Dependencias externas

El proyecto de software fue realizado bajo el lenguaje de Programación C, el cual tiene diversas librerías integradas de las que se hizo uso para poder realizar las implementaciones de los distintos módulos y TDAs.

I. Librerías empleadas a lo largo del proyecto

1. *stdio.h*[3] (cabecera estándar E/S), es el archivo de cabecera que contiene las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones, estándar, de entrada y salida, así como la definición de tipos necesarios para dichas operaciones. Las funciones, macros y variables de biblioteca a lo largo del proyecto son:

- **Variables de biblioteca:**
  - **FILE:** Tipo de objeto para almacenar información para un flujo de archivos.
- **Macros:**
  - **EOF:** Esta macro es un entero negativo que permite indicar que se ha llegado al final de la lectura del archivo.
- **Funciones:**
  - **fclose(FILE *f):** Cierra el acceso al archivo, vaciando todos los búferes.
  - **feof(FILE *f): int:** Verifica si se ha activado el indicador de fin de archivo o no, y devuelve un número distinto de cero en caso positivo, o 0 en caso negativo.
  - **fopen(char * path_archivo, char * modo): FILE *:** Abre el archivo indicado por el path_archivo en el modo indicado por el usuario (puede ser w-escritura, r-Lectura), y devuelve el puntero al archivo o NULL si no se pudo abrir el archivo.
  - **fprint(FILE * f, char * formato, char * parametro_1, ..., char * parametro_n): int**: Envía la salida formateada a un archivo, esto es, escribe sobre un archivo.
  - **printf(char * formato, char * parametro_1, ..., char * parametro_n): int:** Muestra en pantalla la salida formateada para un conjunto de parámetros.
  - **fgetc(FILE * f): int**: Realiza la lectura del siguiente carácter del archivo y lo retorna como un entero (si se lo almacena en una variable char, se castea y recupera el char correspondiente al código ASCII).

[3] stdio.h: https://tutoriales.edu.lat/pub/c-standard-library/stdio-h/biblioteca-c-stdio-h

2. stlib.h[4] es el archivo de cabecera de la biblioteca estándar de propósito general del lenguaje de programación C. Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras. Las funciones, macros y variables de biblioteca empleadas a lo largo del proyecto son:

- **Macros:**
    - **NULL:** Es el valor constante de un puntero nulo.
- **Funciones:**
    - **free(void * ptr):** Libera la memoria previamente reservada por una llamada malloc (reservación de memoria).
    - **malloc(int size): void *:** Reserva la cantidad de memoria indicada por **size** y retorna un puntero.
    - **exit(int value):** Terminar ejecución del programa. Utilizado para finalizar la ejecución en caso extremos de error.

3. string.h[5] es la biblioteca de manejo de cadenas provisto por C y contiene un conjunto de funciones para manipular cadenas, ya sea para copiar, cambiar caracteres, comparar cadenas, entre otras funciones. Las funciones, macros y variables de biblioteca empleadas a lo largo del proyecto son:

- **Funciones:**
    - **strcat(char * destino, char * origen): char *:** Agrega la cadena apuntada por el origen a lo apuntado por destino.
    - **strcmp(char * str1, char * str2): int:** Realiza la comparación de dos cadenas de caracteres y devuelve 0 si son iguales, entero positivo si str1 es mayor a str2 y entero negativo en caso contrario.
    - **strcpy(char * destino, char * origen): char *:** Copia la cadena apuntada por origen a destino y retorna el resultado (y afecta al puntero destino).
    - **strlen(char * str): size_t:** Devuelve un entero que indica la longitud de la cadena hasta el carácter nulo pero sin contar a este último.

[4] stlib.h: https://tutoriales.edu.lat/pub/c-standard-library/stdlib-h/biblioteca-c-stdlib-h
[5] string.h: https://tutoriales.edu.lat/pub/c-standard-library/string-h/biblioteca-c-string-h

4. dirent.h[6] es la cabecera de la biblioteca POSIX de C para el lenguaje de programación C que contiene construcciones que facilitan el recorrido de directorios. La función no forma parte del estándar C, pero se considera "pseudoestándar" y suele ser portable entre plataformas. Las funciones, macros y variables de biblioteca empleadas a lo largo del proyecto son:

- **Miembros:**
  - **struct dirent:** Una estructura que incluye los siguientes miembros:
    - **d_ino: ino_t:** Número de serie del archivo.
    - **d_name[]: char*:** Nombre de la entrada (archivo). Utilizado para devolver el nombre del archivo o directorio interno.
- **Funciones:**
  - **closedir(DIR* dirp): int:** Cierra el flujo de directorios referenciado por dirp. Al retornar, dirp ya no puede apuntar a un objeto accesible del tipo DIR. Si se completa con éxito, closedir() devuelve 0. En caso contrario, devuelve -1 y errno indica el error.
  - **opendir(char * directorio): DIR*:** Abre un flujo de directorio correspondiente al directorio dado. El flujo de directorio se posiciona en la primera entrada. Si se completa con éxito, opendir() devuelve un puntero a un objeto de tipo DIR. En caso contrario, devuelve un puntero nulo y errno indica el error.
  - **readdir(DIR* dirp): struct dirent*:** Devuelve un puntero a una estructura que representa la entrada de directorio en la posición actual en el flujo de directorio especificado por el argumento dirp, y posiciona el flujo de directorio en la siguiente entrada. Devuelve un puntero nulo al llegar al final del flujo de directorios. Cuando se encuentra el final del directorio, se devuelve un puntero nulo y errno no se modifica.
  - **rewinddir(DIR * dirp):** Restablece la posición del flujo de directorios al que dirp hace referencia al principio del directorio. También hace que el flujo de directorios se refiera al estado actual del directorio correspondiente, como habría hecho una llamada a opendir().

[6] dirent.h: https://en.wikibooks.org/wiki/C_Programming/POSIX_Reference/dirent.h

II. Módulo TDA Lista

El Módulo Lista hace uso de las siguientes funciones macros, variables de bibliotecas y funciones previamente mencionadas:

De la biblioteca **stlib.h**, se utiliza los siguientes recursos:

- **Macros:**
  - **NULL:** Para establecer una referencia nula o comprobar por una referencia nula.
- **Funciones:**
  - **exit(int value):** Para finalizar la ejecución del programa por causa de algún error grave.
  - **free(void * ptr):** Para liberar la memoria reservada por el uso de un malloc.
  - **malloc(int size):** Para reservar memoria dinámica para crear una lista, un nodo o un elemento.

De la biblioteca **stdio.h**, se utiliza los siguientes recursos:

- **Funciones:**
  - **printf():** Para mostrar los mensajes de error ocurridos en la lista.

III. Módulo TDA Multiset

EL Módulo Multiset hace uso de las siguientes funciones macros, variables de bibliotecas y funciones previamente mencionadas:

De la biblioteca **stlib.h**, se utiliza los siguientes recursos:

- **Macros:**
  - **NULL:** Para establecer una referencia nula o comprobar por una referencia nula.
- **Funciones:**
  - **exit(int value):** Para finalizar la ejecución del programa por causa de algún error grave.
  - **free(void * ptr):** Para liberar la memoria reservada por el uso de un malloc.
  - **malloc(int size):** Para reservar memoria dinámica para crear un multiset.

De la biblioteca **stdio.h**, se utiliza los siguientes recursos:

- **Funciones:**
  - **printf():** Para mostrar los mensajes de error ocurridos en la lista.

IV. Módulo Cuentapalabras

EL Módulo Cuentapalabras hace uso de las siguientes funciones macros, variables de bibliotecas y funciones previamente mencionadas:

De la biblioteca **stlib.h**, se utiliza los siguientes recursos:

- **Macros:**
  - **NULL:** Para establecer una referencia nula o comprobar por una referencia nula.
- **Funciones:**
  - **exit(int value):** Para finalizar la ejecución del programa por causa de algún error grave.
  - **free(void * ptr):** Para liberar la memoria reservada por el uso de un malloc.
  - **malloc(int size):** Para reservar memoria dinámica para crear una lista, un nodo o un elemento.

De la biblioteca **stdio.h**, se utiliza los siguientes recursos:

- **Variables de biblioteca:**
  - **FILE:** Para abrir los archivos y poder leer o escribir sobre ellos.
- **Macros:**
  - **EOF:** Para detectar si se ha llegado al final del archivo.
- **Funciones:**
  - **fclose(FILE *f):** Para cerrar un archivo previamente abierto que ya no se utiliza o que se terminó de escribir.
  - **feof(FILE *f): int:** Para verificar que aún no se ha llegado al final del archivo.
  - **fopen(char * path_archivo, char * modo): FILE *:** Para abrir el flujo de un archivo en un modo de lectura o escritura.
  - **fprint(FILE * f, char * formato, char * parametro_1, ..., char * parametro_n): int**: Para escribir sobre los archivos de salida los datos almacenados en los distintos multisets.
  - **printf(char * formato, char * parametro_1, ..., char * parametro_n): int:** Para mostrar los mensajes de bienvenida, de error y los distintos resultados obtenidos de la ejecución del programa.
  - **fgetc(FILE * f): int**: Para recuperar el siguiente caracter de un flujo de archivo abierto. Esto se utilizó principalmente para recopilar las palabras de a un caracter, para poder identificar los caracteres que dividen palabras.

De la biblioteca **string.h**, se utiliza los siguientes recursos:

- **Macros:**
  - **NULL:** Para establecer una referencia nula o comprobar por una referencia nula.
- **Funciones:**
  - **strcat(char * destino, char * origen): char *:** Para concatenar el directorio dado por argumento al programa con los distintos nombres de archivos de texto hallados.
  - **strcmp(char * str1, char * str2): int:** Para realizar la comparación de la función de ordenamiento y poder determinar el orden ascendente en caso de igualdad de repeticiones mediante la comparación de las cadenas.
  - **strcpy(char * destino, char * origen): char *:** Para copiar el nombre del archivo de texto encontrado en el directorio al arreglo de nombre de archivos de textos.
  - **strlen(char * str): size_t:** Para contabilizar la longitud de una cadena de caracteres.

De la biblioteca **dirent.h**, se utiliza los siguientes recursos:

- **Miembros:**
  - De **struct dirent** se utilizó el atributo **d_name[]: char*** para obtener el nombre del elemento que se estaba recorriendo en el directorio.
- **Funciones:**
  - **closedir(DIR* dirp): int:** Para cerrar el flujo del directorio.
  - **opendir(char * directorio): DIR*:** Para abrir un flujo al directorio dado por argumento al programa y así poder verificar si existe dicho directorio o no.
  - **readdir(DIR* dirp): struct dirent*:** Recorrer el directorio en busca de los archivos de texto.
  - **rewinddir(DIR * dirp):** Restablecer el flujo del directorio al inicio del mismo (equivalente a un opendir) y así poder recomenzar a leer el directorio.

3. Descripción de procesos y servicios

3.1. Servicio ofrecido por el sistema

El presente sistema de software ***requiere*** que sea invocado indicando el directorio de archivos sobre el cual se recopilarán todos los archivos de textos (ver 3.2. Invocación del programa).

El sistema ***tiene una única función*** en concreto, y esta es que, en base al directorio, se recuperan todos los archivos de textos para posteriormente leerlos y contabilizar las repeticiones de cada palabra de acuerdo a lo definido en 1.1. Definición general del Proyecto de software, plasmando dicha información mediante la estructura provista por el TDA Multiset.

Luego, se procede a recuperar los elementos de cada multiset empleando alguna implementación del TDA Lista, a lo que los elementos luego son ordenados de manera ***ascendente*** según la cantidad de repeticiones y en caso de igualdad, se ordenan ascendentemente por orden de la palabra.

Finalmente, el mismo sistema procede a construir los archivos de acuerdo a lo especificado en 1.1. Definición general del Proyecto de software, comunicando al usuario que se han construido dichos archivos con éxito.

3.2. Instalación e Invocación del programa

I. Instalación del programa

El proyecto de software es presentado como un conjunto de archivos de código y encabezados, los cuales deben ser compilados para construir el ejecutable en cuestión.

Prerrequisitos: Se requiere de la instalación del compilador de C/C++, el cual puede ser descargado desde el siguiente link y a la hora de elegir qué componente instalar, se elige el compilador de C (puede aparecer como Objective-C).

Instalación: Más que instalación, es un proceso de compilación del código fuente del proyecto que permite obtener un programa ejecutable. Para esto, se debe seguir el siguiente instructivo para ***generar un programa ejecutable*** y así poder utilizarlo:

1. Abrir el intérprete de comandos y dirigirse a la carpeta donde está ubicado los archivos fuente del sistema (para esto se puede utilizar la instrucción CD para moverse de un directorio a otro).
    - Antes de seguir, se debe verificar que ***gcc.exe*** sea un comando ejecutable desde cualquier parte en el intérprete de comandos.
    - Si no fuera así, se debe agregar a las ***variables del entorno*** el PATH (la ruta) hacia dicho programa, que está ubicado, por lo general, en ***MinGW\bin*** (de acuerdo a la instalación realizada de MinGw, es donde estará ubicada, por caso, al instalar

Code::blocks con MinGW, esto se encuentra dentro de la carpeta de Code::blocks en ***C:\Program Files\CodeBlocks\MinGW***).

2. Ahora es necesario construir los archivos objeto de cada archivo de código (los archivos .c), para eso se emplean las siguientes líneas:
    - ***gcc.exe -Wall -c cuentapalabras.c -o cuentapalabras.o***
    - ***gcc.exe -Wall -c lista.c -o lista.o***
    - ***gcc.exe -Wall -c mutiset.c -o multiset.o***
3. Generados los archivos objetos, ahora se procede a construir el archivo ejecutable (.exe), para esto se emplea la siguiente línea en el intérprete de comandos:
    - ***gcc.exe -o cuentapalabras.exe cuentapalabras.o lista.o multiset.o***
4. Si todo ha salido bien, entonces se ha de generar el archivo *cuentapalabras.exe* y está listo para ser utilizado.

II. Invocación del programa

La invocación del programa se realiza también por línea de comandos (o intérprete de comandos), para eso es necesario seguir el siguiente instructivo para hacerlo correctamente.

1. Abrir el intérprete de comandos y dirigirse a la carpeta donde está el archivo ejecutable construido en I. Instalación del programa.
2. Luego, invocar al programa empleando la siguiente instrucción

    cuentapalabras.exe -h [directorio]

    - donde ***[directorio]*** es un directorio existente del sistema de archivos de la computadora.
3. Si todo ha salido bien, se habrá leído los archivos de textos del directorio dado y generados los dos archivos de conteo de palabras en dicho directorio.

III. Posibles errores de invocación

El archivo ejecutable debe ser invocado de la manera dada en II. Invocación del programa, sin embargo, si no se respeta esto puede ocurrir algunos de los siguientes errores.

1. ***Sin parámetro o directorio.*** No se ha pasado el parámetro -h o el directorio a analizar por parámetro. Lo que finaliza con el programa indicando cuál es el formato de invocación del programa. No genera ningún archivo de salida.
2. ***Directorio erróneo o mal tipeado.*** Si el directorio no es válido, esto es, que no existe o por cualquier motivo sea no se puede acceder, entonces el programa finaliza por no poder haber abierto dicho directorio (puesto que es inaccesible o no existe).

3. ***Directorio sin archivos de texto.*** Si un directorio dado es válido pero no tiene archivos de textos que leer, entonces no se genera ningún archivo de salida, ya que se asume que no tiene sentido el hecho de generar dichos archivos.

IV. Posibles errores en ejecución

Suponiendo una correcta invocación del programa ejecutable, en casos muy puntuales puede ocurrir que se produzcan errores en la ejecución del programa. El error más grave que puede ocurrir es el siguiente:

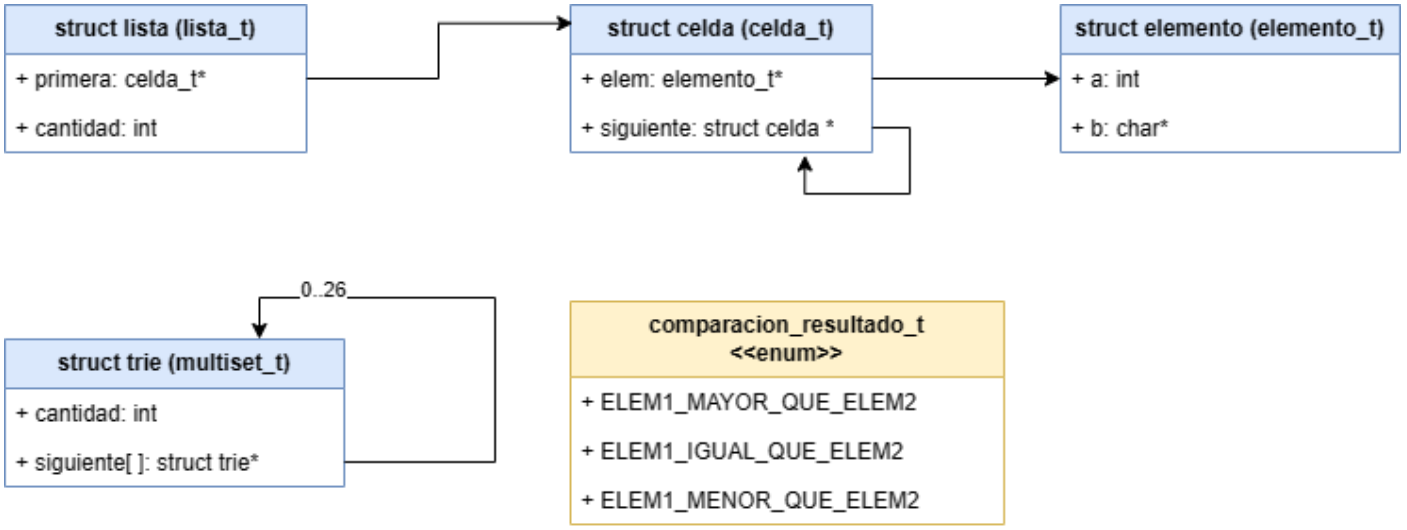
1. ***Reservación de memoria no realizada.*** Para almacenar los datos en memoria, el software le pide al sistema operativo que le brinde memoria suficiente para almacenar dichos datos. Cuando no se logra tal propósito, se tiene que no se ha reservado memoria para los datos y por ende, el sistema finaliza indicando el error.

Los posibles errores en ejecución (mostrados por el programa) son los siguientes:

- ***ERROR_LISTA_BUSQUEDA (-1):*** Error que aparece si la búsqueda de un producto no se efectua con éxito, por lo general, por dar una posición errónea.
- ***ERROR_LISTA_ELIMINAR (-2):*** Error que surge al eliminar un elemento de una posición no válida.
- ***ERROR_LISTA_MEMORIA (-3):*** Error que surge cuando no se logra reservar memoria para crear la lista.
- ***ERROR_MULTISET_MEMORIA (-4):*** Error que surge cuando no se logra reservar memoria para crear el multiset.
- ***ERROR_ELEMENTO_MEMORIA (-5):*** Error que surge al no poder reservar memoria para almacenar un elemento.
- ***ERROR_CUENTAPALABRAS_CONTADOR (-6):*** Error que surge cuando no se logra reservar memoria para el contador de archivos.
- ***ERROR_CUENTAPALABRAS_APERTURA_ARCHIVO (-7)***: Error que surge cuando no se puede abrir un archivo de texto.
- ***ERROR_CUENTAPALABRAS_CREACION_ARCHIVO_SALIDA (- 8):*** Error que surge cuando no es posible construir los archivos de salida por algún motivo (por ejemplo, permisos de escritura).
- ***ERROR_CUENTAPALABRAS_MEMORIA (-9):*** Error que surge por no poder reservar memoria para la creación de una cadena de caracteres.
- ***ERROR_CUENTAPALABRAS_APERTURA_DIRECTORIO (-10):*** Error que surge por no poder abrir el directorio dado, puesto que el mismo está mal escrito, esto es, no existe.

3.3. Diagrama de flujos del sistema

En el diagrama de flujo que se muestra a continuación se puede observar cómo están conectadas las siguientes estructuras (struct) empleadas en el proyecto de software.



3.4. Documentación de las estructuras de datos

La información de esta sección ha sido recolectada empleando la herramienta DoxyGen, la cual permite generar documentación para cualquier lenguaje de programación.

I. Referencia de la estructura celda

La ***celda*** permite almacenar un elemento 'elem' y una referencia a la siguiente celda.

Campos de datos ***elemento_t * elem*** ***struct celda * siguiente***
Descripción detallada La celda permite almacenar un elemento 'elem' y una referencia a la siguiente celda.
La documentación para esta estructura fue generada a partir del siguiente fichero: ***lista.c***

II. Referencia de la estructura elemento

Modela un ***elemento*** que tendrá como valor a la cantidad de repeticiones de una palabra y el valor b será un puntero a la palabra en cuestión.

Campos de datos ***int a*** ***char * b***
Descripción detallada Modela un elemento que tendrá como valor a la cantidad de repeticiones de una palabra y el valor b será un puntero a la palabra en cuestión.
La documentación para esta estructura fue generada a partir del siguiente fichero: ***lista.h***

III. Referencia de la estructura lista

Modela una ***lista*** simplemente enlazada y sin nodo centinela.

Campos de datos ***celda_t * primera*** *//Puntero a la primera celda de la lista.* ***int cantidad***
Descripción detallada Modela una lista simplemente enlazada y sin nodo centinela.
La documentación para esta estructura fue generada a partir del siguiente fichero: ***lista.h***

III. Referencia de la estructura trie

Modela un ***árbol trie***, donde el rótulo será la cantidad de repeticiones de la palabra hasta el nodo actual y puede tener hasta 26 nodos hijos, donde cada hijo representa un carácter entre 'a' y 'z' (excluyendo a la ñ).

Campos de datos ***int cantidad*** *//Cantidad de repeticiones.* ***struct trie * siguiente [26]*** *//26 nodos hijos que representan a una letra de la a a la z.*
Descripción detallada Modela un árbol trie, donde el rótulo será la cantidad de repeticiones de la palabra hasta el nodo actual y puede tener hasta 26 nodos hijos, donde cada hijo representa un carácter entre 'a' y 'z' (excluyendo a la ñ). Representa un árbol Trie, donde cada nodo representa un carácter distinto.
La documentación para esta estructura fue generada a partir del siguiente fichero: ***multiset.c***

3.5. Documentación de los archivos

La información de esta sección ha sido recolectada empleando la herramienta DoxyGen, la cual permite generar documentación para cualquier lenguaje de programación.

I. Operaciones del TDA Lista

Implementación del TDA Lista. Implementación del TDA Lista. El TDA LISTA almacenará elementos pares de la forma <*cadena_de_caracteres, entero*>. La lista es una lista sin centinela con posición indirecta.

Librerías incluidas en el archivo:

- #include <stdio.h>
- #include <stdlib.h>
- #include "define.h"
- #include "lista.h"

Definiciones de constantes (definidas en lista.h)

- #define ***ERROR_LISTA_BUSQUEDA*** -1
- #define ***ERROR_LISTA_ELIMINAR*** -2
- #define ***ERROR_LISTA_MEMORIA*** -3

Funciones

celda_t * aux_construir_celda_nueva (elemento_t * e)

Construye una nueva celda con el elemento el contenido en dicha celda y celda siguiente en NULL.

Parámetros

- **e**: Puntero al elemento.

Excepciones

- ERROR_LISTA_MEMORIA: Si no se puedo reservar memoria para una celda o elemento.

Devuelve: Una celda con el elemento contenido.

int aux_es_archivo_txt (char * name)

Comprueba si un puntero a una cadena de caracteres es el nombre de un archivo con extensión .txt.

Parámetros

- **name:** Puntero a cadena de caracteres.

Devuelve: TRUE si es un archivo .txt y FALSE en caso contrario.

elemento_t * aux_ordenar_lista (celda_t * celda_actual, int pos_inicial, int pos_final, funcion_comparacion_t comparar)

Realiza el procedimiento de ordenar la lista recursivamente.

Parámetros

- **celda_actual**: Puntero a la celda actual.
- **pos_inicial**: Entero positivo con la posicion inicial.
- **pos_final**: Entero positivo con la posicion final.
- **comparar**: Funcion de comparación para realizar el ordenamiento según el criterio que establezca.

Devuelve: Puntero al elemento con el elemento ordenado. Si se llegó al final. Si no llegó al final.

unsigned int lista_cantidad (lista_t * l)

Devuelve la cantidad de elementos de la lista 'l'.

Parámetros

- **l**: Puntero a la lista de elementos.

Devuelve: Entero positivo mayor o igual a 0.

lista_t * lista_crear ()

Crea una lista vacía y la devuelve.

Excepciones

- ERROR_LISTA_MEMORIA: si no se logra reservar memoria para la lista.

Devuelve: Puntero a una lista con memoria reservada.

elemento_t * lista_elemento (lista_t * l, unsigned int pos)

Devuelve un puntero al elemento que ocupa la posición 'pos' de la lista.

Parámetros

- **l**: Puntero a la lista de elemntos.
- **pos**: Entero positivo que indica la posicion (0<pos<n, donde n es el tamaño de la lista).

Devuelve: El elemento de la posición 'pos' (esto es, para una lista de n elemntos, pos<n), de lo contrario, retorna NULL.

elemento_t * lista_eliminar (lista_t * l, unsigned int pos)

Elimina el elemento de la posición 'pos' de la lista y lo retorna.

Parámetros

- **l**: Puntero a la lista de elementos.
- **pos**: Entero positivo que indica la posicion a eliminar (0<pos<n, donde n es el tamaño de la lista).

Devuelve: El elemento removido de la lista en caso de que la posición pertenezca a la lista (esto es, para una lista de n elemntos, pos<n), de lo contrario, retorna NULL.

int lista_insertar (lista_t * l, elemento_t elem, unsigned int pos)

Inserta el elemento 'elem' en la posición 'pos' de la lista. Si la posición es menor o igual, entonces se procede a insertar el elemento. Se construye la nueva celda con el elemento en cuestión.

- Caso 1: Lista vacía
- Caso 2: Lista con elementos y posición=0 (al inicio de la lista)
- Caso 3: Insertar al final de la lista
- Caso 4: Otra posición tal que 0<=pos<long_lista

Parámetros

- **l**: Puntero a la lista de elementos.
- **elem**: Elemento a insertar en la lista.
- **pos**: Entero positivo que indica la posicion de inserción (0<pos<n, donde n es el tamaño de la lista).

Excepciones

- ERROR_LISTA_MEMORIA: si no se reserva memoria.
- ERROR_LISTA_BUSQUEDA: si se pasa una posicion fuera de rango.

Devuelve: TRUE si se inserta el elemento correctamente.

int lista_ordenar (lista_t * l, funcion_comparacion_t comparar)

Dada la lista 'l' y la función 'comparar' ordena la lista de acuerdo al criterio de dicha función.

Parámetros

- **l**: Puntero a la lista de elementos.

- **comparar**: Función de comparación de elementos.

Devuelve: TRUE si la lista fue ordenada con exito, de lo contrario, false.

int lista_vacia (lista_t lista)

Devuelve verdadero (distinto de 0) si la lista está vacía, y falso (igual a 0) si la lista contiene al menos un elemento.

Parámetros

- **lista**: Lista de elementos.

Devuelve: TRUE si la lista está vacía, de lo contrario, FALSE.

II. Operaciones del TDA Multiset

Implementación del TDA Cuenta Palabras. Modela las operaciones necesarias para un Multiset, que es una colección de elementos sin orden establecido que acepta elementos repetidos.

Librerías incluidas en el archivo:

- #include <stdio.h>
- #include <stdlib.h>
- #include "multiset.h"
- #include "lista.h"
- #include "define.h"

Definiciones de constantes

- #define ***ERROR_MULTISET_MEMORIA*** -4
- #define ***ERROR_ELEMENTO_MEMORIA*** -7

Funciones

void aux_cargar_elementos_en_lista (lista_t * L, struct trie * T, char s[], int length_s)

Dado el árbol T, realiza la carga de los elementos cuya cantidad de repeticiones es mayor o igual a 1 en la lista L.

Parámetros

- **L**: Puntero a la lista.
- **T:** Puntero a la estructura del árbol trie.

- **s:** Arreglo de caracteres de la iteración previa.
- **length_s**: Longitud del arreglo previo.

Excepciones

- ERROR_ELEMENTO_MEMORIA: si no se pudo reservar memoria para el elemento o para su contenido.

elemento_t aux_construir_elemento (int cant_repeticiones, char * s, int length_s)

Construye un elemento para la cadena de caracteres y la cantidad de repeticiones dada..

Parámetros

- cant_repticiones: Entero mayor o igual a 1.
- s: Puntero a una secuencia de caracteres.
- length_s: Entero mayor o igual a 1 que representa la longitud de la cadena s.

Excepciones

- ERROR_ELEMENTO_MEMORIA: si no se pudo reservar memoria para el elemento o para su contenido.

Devuelve: Elemento construido y con memoria reservada.

void aux_liberar_memoria (struct trie * nodo)

Realiza la liberación de la memoria, esto es, remueve todo dato perteneciente al nodo de la memoria.

Parámetros

- **nodo**: Puntero a un nodo del árbol.

void aux_multiset_eliminar (struct trie * nodo)

Elimina los nodos del árbol de manera recursiva.

Parámetros

- **nodo**: Puntero a un nodo del árbol.

char aux_recuperar_caracter_en_posicion (int pos)

Dada una posición que corresponde a un nodo del árbol (entre 0 y 25), se recupera el caracter que simboliza dicha posición.

Parámetros

- **pos**: Número entero entre 0 y 25.

Devuelve: Carácter entre 'a' y 'z' (excluyendo a ñ).

int aux_recuperar_posicion_en_alfabeto (char * ch)

Dado un char, devuelve la posición del índice entre 0 y 25 del nodo trie que le corresponde al char.

Parámetros

- **ch**: Puntero al carácter.

Devuelve: Entero entre 0 y 25 si el char está entre 'a' y 'z' (excluyendo ñ). En caso contrario, devuelve -1.

int multiset_cantidad (multiset_t * m, char s[])

Devuelve la cantidad de repeticiones de la palabra 's' en el multiset m

Parámetros

- **m**: Puntero al multiset.
- **s:** Puntero al inicio de la cadena de caracteres.

Devuelve: Entero positivo mayor a 0 si se hay repticiones de s, en cambio, si no está definida o la misma no tiene repeticiones devuelve 0.

multiset_t * multiset_crear ()

Crea un multiset vacio de palabras y lo devuelve.

Excepciones

- ERROR_MULTISET_MEMORIA: si el programa no logra reservar memoria para el multiset.

Devuelve: Puntero al multiset construido o NULL en caso de error.

lista_t multiset_elementos (multiset_t * m, int(*)(elemento_t, elemento_t) f)

Devuelve una lista de tipo lista_t ordenada según la función 'f' con todos los elementos del multiset 'm' y la cantidad de apariciones de cada uno.

Parámetros

- **m:** Puntero al multiset.
- **f:** Función de comparación de elementos.

Devuelve: Lista de elementos ordenados con las palabras y su respectiva cantidad de repeticiones.

void multiset_eliminar (multiset_t ** m)

Elimina el multiset 'm' liberando el espacio de memoria reservado. Luego de la invocación 'm' debe NULL.

Parámetros

- **m**: Puntero a puntero de multiset.

void multiset_insertar (multiset_t * m, char * s)

Inserta la palabra 's' al multiset 'm'. Si la reserva de memoria no se realiza correctamente, puede finalizar la ejecución del programa con ERROR_MULTISET_MEMORIA.

Parámetros

- **m**: Puntero al multiset.
- **s**: Puntero al inicio de la cadena de caracteres.

Excepciones

- ERROR_MULTISET_MEMORIA: si no se pudo crear el multiset.
- ERROR_ELEMENTO_MEMORIA: si no se pudo reservar memoria para el elemento o para su contenido.

III. Operaciones del Cuentapalabras

Implementación del TDA Cuenta Palabras. Modela las operaciones necesarias para poder leer un directorio de archivos .txt y poder contabilizar la cantidad de palabras que hay, y posteriormente plasmar dicha información en archivos de salida.

Librerías incluidas en el archivo:

- #include <stdio.h>
- #include <stdlib.h>
- #include <string.h>
- #include <dirent.h>
- #include "multiset.h"
- #include "lista.h"

Definiciones de constantes

- #define **ERROR_CUENTAPALABRAS_CONTADOR** -6
- #define **ERROR_CUENTAPALABRAS_APERTURA_ARCHIVO** -7
- #define **ERROR_CUENTAPALABRAS_CREACION_ARCHIVO_SALIDA** - 8
- #define **ERROR_CUENTAPALABRAS_MEMORIA** -9
- #define **ERROR_CUENTAPALABRAS_APERTURA_DIRECTORIO** -10

Funciones

multiset_t * aux_cargar_multiset (char * path, multiset_t * m_total)

De acuerdo a la ruta al archivo dada, se procede a leer el archivo de texto y se recopila cada palabra y se las contabiliza.

Parámetros

- path: Puntero a cadena de caracteres que conforman la ruta hacia el archivo a leer.
- m_total: Multiset donde se cargarán las palabras leidas en el documento.

Excepciones

- ERROR_CUENTAPALABRAS_APERTURA_ARCHIVO: si no se pudo abrir el archivo.

Devuelve: Multiset con las palabras contadas pertenecientes al archivo dado.

int aux_es_archivo_txt (char * name)

Comprueba si un puntero a una cadena de caracteres es el nombre de un archivo con extensión .txt.

Parámetros

- **name:** Puntero a cadena de caracteres.

Devuelve: TRUE si es un archivo .txt y FALSE en caso contrario.

int aux_es_palabra_sin_caracteres_especiales (char * ch)

Comprueba si es una palabra sin caracteres especiales, esto es, solo está compuesta por letras del alfabeto.

Parámetros

- **ch:** Puntero a cadena de caracteres.

Devuelve: TRUE si es una cadena sin caracteres especiales, de lo contrario, retorna FALSE.

void aux_exportar_multiset_a_archivo (FILE * file, char * nombre_archivo, multiset_t * multiset_archivo)

Escribe en el archivo indicado el nombre del archivo seguido del contenido del multiset.

Parámetros

- **file:** Puntero al manejador de archivo. Requiere que esté abierto el archivo para poder ser escrito.
- **nombre_archivo:** Puntero a cadena de caracteres que conforman el nombre del archivo.
- **multiset_archivo:** Puntero a multiset de palabras ordenadas.

void aux_liberar_memoria_elemento (elemento_t * elem)

Libera la memoria reservada por el elemento en cuestión.

Parámetros

- **elem:** Puntero a elemento.

char * aux_recuperar_cadena (FILE * f, int longitud_cadena)

Recorre recursivamente la próxima cadena a recuperar por del archivo y la devuelve.

Parámetros

- **f:** Puntero a archivo.
- **longitud_cadena:** Longitud de la cadena a leer.

Excepciones

- ERROR_CUENTAPALABRAS_MEMORIA: Si no se logró reservar memoria para la cadena.

Devuelve: Puntero a cadena creada o NULL si la cadena en cuestión en un caracter separador de palabras.

DIR * cuentapalabras_abrir_directorio (char * path)

Abre el directorio dado por la ruta path y retorna su controlador.

Parámetros

- **path:** Puntero a cadena de caracteres que representa el directorio.

Devuelve: Puntero al controlador de directorio o NULL si el directorio dado no es válido.

void cuentapalabras_construir_archivos_salida (char * directorio, char ** nombre_archivo, int cant_filas)

Realiza la construcción de los archivos cadauno.out y totales.out en base a los archivos de textos encontrados en el directorio dado. Importante: Los mencionados archivos a construir se escribirán en el directorio dado.

Parámetros

- **directorio**: Puntero a cadena de caracteres que representa el directorio.
- **nombre_archivo**: Puntero a punteros de cadenas de caracteres que representan los nombres de los archivos.
- **cant_filas**: Entero que indica la cantidad de archivos de textos a leer.

Excepciones

- ERROR_CUENTAPALABRAS_MEMORIA Si no es reservada memoria para la creación del puntero al puntero de un multiset.
- ERROR_CUENTAPALABRAS_CREACION_ARCHIVO_SALIDA Si no se pudo crear los archivos cadauno.out o totales.out.

void cuentapalabras_liberar_memoria_nombres_archivos (char ** C, int cant_filas)

Libera la memoria reservada para los nombres de los archivos recuperados.

Parámetros

- **C:** Puntero a punteros de cadenas de caracteres.
- **cant_filas**: Entero positivo que representa la cantidad de archivos a eliminar.

char ** cuentapalabras_recopilar_nombres_archivos_txt (DIR * d, int * cant_filas)

Dado un controlador del directorio, se recopila todos los archivos de textos existentes en dicho directorio y se retorna un puntero a punteros de cadenas de caracteres.

Parámetros

- **d:** Puntero al controlador de directorio.
- **cant_filas**: Puntero a un entero, donde se almacenará la cantidad de archivos de texto encontrados.

Excepciones

- ERROR_CUENTAPALABRAS_APERTURA_ARCHIVO: si no se logra abrir el archivo en cuestion.
- ERROR_CUENTAPALABRAS_MEMORIA: si no se logra reservar memoria para la cadena.

Devuelve: Puntero a punteros de cadenas de caracteres que representan los nombres de los archivos de textos encontrados en el directorio dado.

comparacion_resultado_t funcion_comparacion (elemento_t * elem1, elemento_t * elem2)

Función de comparación de dos elementos que permite identificar cual elemento es mayor, menor o igual que otro.

Parámetros

- **elem1**: Puntero a un elemento_t.
- **elem2**: Puntero a un elemento_t.

Devuelve: ELEM1_MAYOR_QUE_ELEM2 si elem1>elem2, ELEM1_MENOR_QUE_ELEM2 si elem1<elem2 y ELEM1_IGUAL_QUE_ELEM2 si elem1=elem2.

void mostrar_mensaje_archivos_a_analizar (char * directorio, char ** nombre, int cant_nombres)

Imprime un mensaje mostrando los nombres de los archivos de texto a analizar en el directorio dado.

Parámetros

- **directorio**: Puntero a cadena de caracteres que representa el directorio.
- **nombre**: Puntero de punteros de cadenas de caracteres que contiene los nombres de los archivos a analizar.
- **cant_nombres**: Entero que representa la cantidad de archivos a analizar.

void mostrar_mensaje_bienvenida ()

Imprime un mensaje de bienvenida al programa.

3.6. Limitaciones del sistema

En torno a las limitaciones del presente proyecto de software, este último tiene una serie de limitaciones que serán enumeradas a continuación:

- **Palabras válidas.** Se considera como una palabra válida a toda cadena de caracteres que esté compuesta solamente por caracteres alfabéticos, esto es, de la a hasta la z (en minúsculas), exceptuando a la ñ.
- **Invocación del software.** El software solamente puede ser invocado empleando la signatura dada en II. Invocación del programa. Empleando esta plantilla de invocación el software actúa como se ha mencionado anteriormente, en caso contrario, solo muestra un mensaje de cómo debe ser invocado el programa.
- **Procesamiento de archivos.** El software solo procesa los archivos de texto, esto es, los que tenga extensión .txt que se encuentren en el directorio dado.

3.7. Conclusiones

Para finalizar la documentación del presente proyecto de software, y a medida de observación se puede destacar que lo más complejo de la implementación ha sido la implementación del código para recorrer el multiset y recuperar cada una de las palabras con repeticiones.

Luego de ese detalle, la implementación del proyecto ha sido bastante directa, salvo los errores en donde no se almacenaba bien una cadena, puesto que en lugar de poner el caracter nulo **\0** se ponía el carácter **\n** lo que producía que al escribir las cadenas en los archivos se escribirán con “basura”, esto es, caracteres en memoria que no representaban nada.