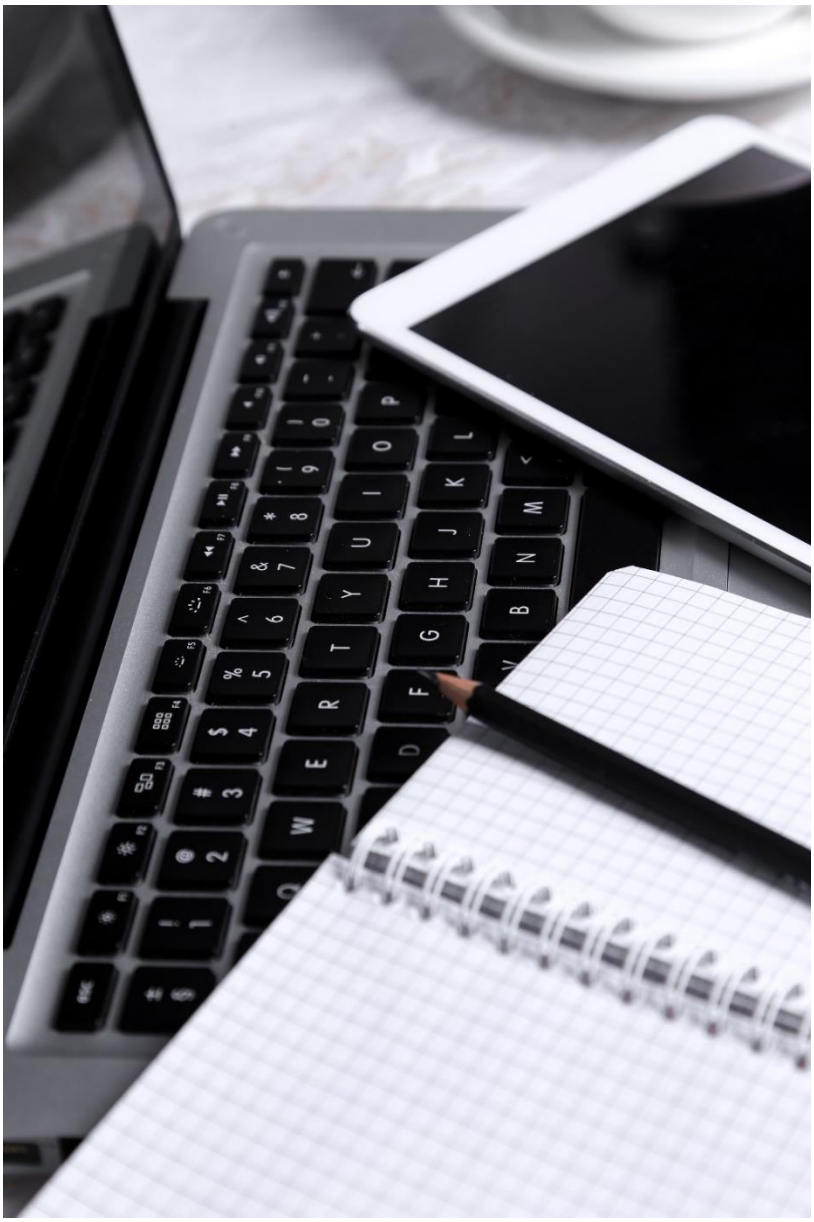
Proyecto de programación en C



Primer cuatrimestre - 2023

Organización de Computadoras

MOYANO – ESCUDERO

Universidad Nacional del Sur

Comisión 13

PROLYGIN GONZÁLEZ, Mayra Ludmila – VERA, Agustín

Definiciones y especificación de requerimientos 3

Definición general del Proyecto de software 3

Especificación de requerimientos del proyecto 3

Requisitos generales 3

Requisitos funcionales 3

Información de autoría y *Legacy* del proyecto 3

Especificaciones de procedimientos 3

Herramientas 3

Planificación 4

Arquitectura del sistema 4

Descripción jerárquica 4

Diagrama de módulos 4

Descripción individual de los módulos 4

listaordenada 4

multiset 5

cuentapalabras 5

Dependencias externas 6

`<stdio.h>` 6

`<stdlib.h>` 6

`<string.h>` 6

`<dirent.h>` 6

`<ctype.h>` 6

Diseño del modelo de datos 6

comparacion_resultado_t 6

elemento_t 6

celda_t 7

Datos de entrada y salida 7

Descripción de procesos y servicios ofrecidos por el sistema 7

Procesos ofrecidos 7

Procesamiento del directorio 7

Ayuda 7

Procesos implementados 7

listaordenada 7

multiset 8

Documentación técnica 9

Conclusiones 9

Definiciones y especificación de requerimientos

Definición general del Proyecto de software

`cuentapalabras` es un programa desarrollado en el lenguaje de programación C creado con el fin de cuantificar la cantidad de palabras contenidas en los archivos de texto pertenecientes a un directorio.

A partir de un directorio especificado, contendiendo archivos de texto, se cuantifican la cantidad de apariciones de cada una de las palabras de los anteriormente mencionados, y se crean dos archivos de texto asociados que contienen el cómputo de la cuantificación según cada archivo, y en la totalidad de estos.

Especificación de requerimientos del proyecto

Los requerimientos del proyecto están explicitados en el enunciado de proyecto de cátedra de Organización de Computadoras (primer cuatrimestre, año 2023).

Requisitos generales

- Respetar los nombres de tipos y encabezados de funciones especificados en el enunciado anteriormente mencionado;
- La invocación del programa `cuentapalabras` debe ser invocado de acuerdo con la siguiente línea de comandos,

  ```
  $ cuentapalabras [-h] [directorio de entrada]
  ```

  donde los parámetros entre corchetes denotan parámetros opcionales;
- Uso cuidadoso y eficiente de la memoria, tanto en la reserva como en la liberación de la misma;
- Respetar el esquema de documentación de proyectos de software.

Requisitos funcionales

A continuación, se destacan las funcionalidades provistas. El usuario en ningún momento es capaz de discriminar cuál de las dos desea ejecutar; sólo puede elegir procesar el directorio o pedir una ayuda para obtener un breve resumen del propósito del programa.

- `cadauno`. Crea un archivo de texto que contiene la cantidad de veces que aparece cada palabra en cada uno de los archivos pertenecientes al directorio.
- `totales`. Crea un archivo de texto que contiene la cantidad de veces que aparece cada palabra en todos los archivos pertenecientes al directorio.

Información de autoría y *Legacy* del proyecto

El proyecto de software `cuentapalabras` no forma parte de desarrollos previos, es de la autoría de los estudiantes que conforman la comisión 13 (VERA FRETES, Mauricio Agustín; PROLYGIN GONZÁLEZ, Mayra Ludmila).

Fue desarrollado y testeado en Windows 10 Home Single Language y ningún otro sistema operativo.

Especificaciones de procedimientos

Herramientas

- El desarrollo fue realizado en el entorno de desarrollo Code::Blocks, en el lenguaje de programación C.

Planificación

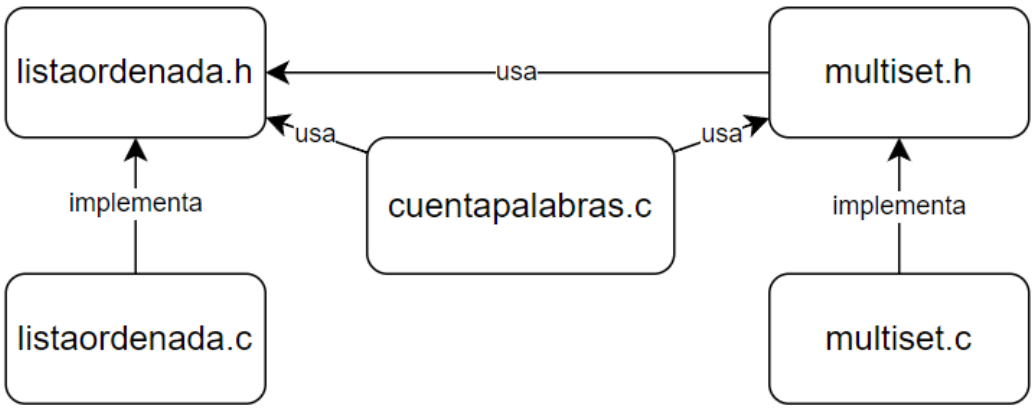
1. El sistema se basa en dos estructuras de datos mayoritariamente independientes; `lista` y `multiset`. Por este motivo el desarrollo fue realizado de forma unitaria y secuencial, respetando el orden antes mencionado.
2. Dados los prototipos requeridos (ver Requisitos generales), y una metodología *divide & conquer*, se resolvieron las funcionalidades de cada uno de los tipos estrucutrados propuestos.
3. El *testing* de las diferentes funcionalidades de las estructuras de datos fue realizado unitariamente, a excepción de la función de encabezado
   `lista_t multiset_elementos(multiset_t *m, int (*f)(elemento_t, elemento_t))`,
   donde era requisito disponer de la estructura lista.
4. Finalmente se desarrolló el correspondiente algoritmo y consecuente código resultante en el programa principal, donde se evaluaron diferentes alternativas de resolución y se optó por el siguiente orden:
   4.1. Evaluar los parámetros especificados por el usuario;
   4.2. Procesar los archivos del directorio especificado por el usuario;
   4.3. Procesar cada uno de los archivos de texto contenidos en el mismo;
   4.4. Delegar el contenido de estos en las funcionalidades propuestas (ver Requisitos funcionales).

Arquitectura del sistema

Descripción jerárquica

Los componentes del sistema se organizan acorde a una estructura monolítica; todos los componentes y operaciones están contenidos dentro de una única unidad. Los datos y las funcionalidades que operan sobre ellos están integrados en una única entidad.

Diagrama de módulos



Descripción individual de los módulos

listaordenada

- Estructura de datos que almacena de forma secuencial un conjunto de elementos de estructura predefinida (entero, cadena de caracteres) a través del enlace simple de celdas
- Este módulo debe de crear nuevas listas ordenadas y ejecutar operaciones sobre ellas.
  - `lista_insertar`. Inserta un elemento en una posición especificada.
  - `lista_eliminar`. Elimina el elemento ubicado en una posición especificada.

    - `lista_elemento`. Devuelve un puntero al elemento ubicado en una posición especificada.
    - `lista_ordenar`. Ordena una lista de acuerdo con un criterio de orden especificado.
    - `lista_cantidad`. Devuelve la cantidad de elementos almacenados.
    - `lista_vacia`. Determina si una lista contiene o no elementos.
- Dado que se trata de una lista ordenada o, al menos, hay posibilidad de establecer un orden entre sus elementos, permite el ordenamiento de las palabras contenidas en los archivos de texto del directorio, de acuerdo a la cantidad de apariciones.
- Para la ejecución exitosa del módulo es necesario que cada vez que se haga uso de una lista, una posición o un elemento, estos deben estar correctamente inidicalizados.
- La definición de los encabezados de este módulo se encuentra en el archivo `listaordenada.h` y, su implementación, en el archivo `listaordenada.c`.

multiset

- Estructura de datos que representa una colección sin orden establecido. Se encuentra implementado con un árbol trie; un árbol de búsqueda o prefijos, donde cada nodo representa un carácter o prefijo de una cadena de caracteres.
- Este módulo debe crear nuevos multisets y ejecutar operaciones sobre ellos.
    - `multiset_insertar`. Inserta una nueva palabra al multiset.
    - `multiset_cantidad`. Devuelve la cantidad de repeticiones de una palabra en un multiset.
    - `multiset_elementos`. Devuelve una lista con todos los elementos del multiset y la cantidad de apariciones de cada uno.
    - `multiset_eliminar`. Elimina al multiset, liberando de forma acorde el espacio de memoria que fue reservado para su creación y múltiples inserciones.
- Dado que almacena una colección con elementos repetidos, permite calcular la cuantificación de la totalidad de palabras contenidas en los archivos de texto del directorio.
- Para la ejecución exitosa del módulo es necesario que cada vez que se haga uso de un multiset debe estar correctamente inicializados. Asimismo, no se consideran caracteres especiales tales como letras con acentos, números, símbolos o la letra ‘Ñ’. Utiliza el archivo `listaordenada.h`.
- La definición de los encabezados de este módulo se encuentra en el archivo `multiset.h` y, su implementación, en el archivo `multiset.c`.

cuentapalabras

- Contiene la implementación de la aplicación de consola consecuente de incluir al método `main()`. Dado un directorio, crea dos archivos de textos que refieren a la cuantificación de palabras existentes en ellos.
- Requiere que el directorio sea un directorio válido y que no se encuentren archivos que no tengan extensión *txt*.
- Utiliza los archivos `listaordenada.h` y `multiset.h`.
- La implementación del código se encuentra en el archivo `cuentapalabras.c`.

Obs.: No es el objetivo de esta sección dar detalles de cómo se realiza la implementación de los módulos, sino únicamente dar una idea general de para qué existe dentro del sistema.

Dependencias externas

`<stdio.h>`

- `printf`. Imprime por pantalla un mensaje con un tipo de formato especificado.
- `scanf`. Lee una entrada.
- `fopen`. Abre un archivo con un nombre parametrizado.
- `fclose`. Cierra un archivo con un nombre parametrizado.
- `feof`. Determina si se llegó al final del archivo.
- `snprintf`. Redirecciona una cadena acorde a lo parametrizado.
- `fprintf`. Escribe en un archivo con un path parametrizado.
- `fscanf`. Lee el contenido de un archivo.

`<stdlib.h>`

- `exit`. Termina la ejecución

`<string.h>`

- `strcmp`. Realiza la comparación entre dos cadenas de caracteres.
- `strcat`. Concatena dos cadenas de caracteres.
- `strlen`. Determina la longitud de una cadena de caracteres.

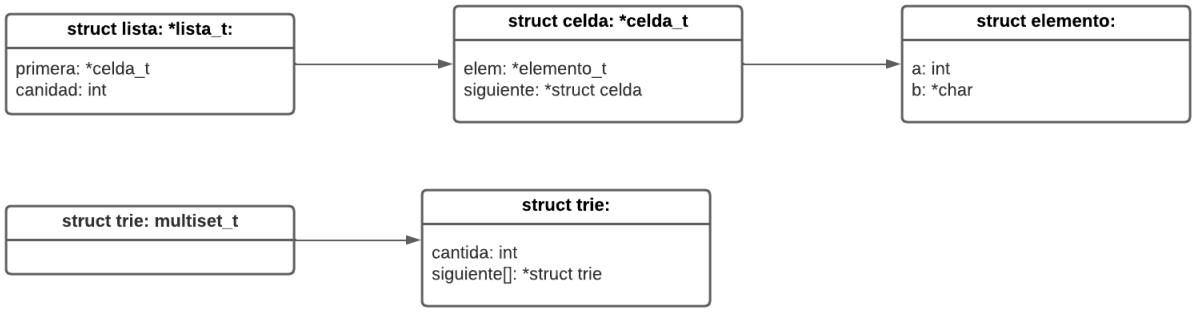
`<dirent.h>`

- `opendir`. Abre un directorio con un path parametrizado.
- `closedir`. Cierra un directorio con un path parametrizado.
- `perror`. Determina la existencia de error en la apertura de un directorio.

`<ctype.h>`

- `tolower`. Computa la letra en minúscula correspondiente a una parametrizada.

Diseño del modelo de datos



Datos internos

comparacion_resultado_t

Tipo de dato de tipo enumerado que establece la relación de orden entre dos elementos; `ELEM1_MAYOR_QUE_ELEM2`, `ELEM1_MENOR_QUE_ELEM2`, `ELEM1_IGUAL_QUE_ELEM2`.

elemento_t

Registro elemento que contiene un entero (`int a`) y una cadena de caracteres (`char *b`).

celda_t

Registro celda que contiene un puntero a un elemento (`elemento_t *elem`) y un puntero a una celda siguiente (`struct celda *siguiente`).

Datos de entrada y salida

- `lista_t`. Puntero a celda; es la primera de la lista.
- `elemento_t`. Puntero a un dato de tipo elemento.
- `función_comparacion_t`. Enumerado que representa relación de orden entre elementos.
- `multiset_m`. Puntero a trie; el nodo raíz del conjunto no ordenado.

Descripción de procesos y servicios ofrecidos por el sistema

Procesos ofrecidos

Procesamiento del directorio

Crea dos archivos de texto que contienen la cuantificación de las palabras existentes en los archivos de texto contenidos en el directorio.

1. Abrir directorio
   1.1. Si no es posible, finalizar ejecución.
   1.2. Caso contrario, guardar en una lista los nombres de los archivos almacenados en el mismo.
2. Crear un multiset para todos los archivos.
3. Por cada uno de los archivos de texto contenidos en el directorio hacer:
   3.1. Procesar archivo.
      3.1.1. Abrir el archivo.
      3.1.2. Crear un multiset para el archivo corriente.
      3.1.3. Leer línea a línea.
      3.1.4. Guardar cada línea en los dos multiset.
      3.1.5. Retornar multiset.
   3.2. `cada_uno`.
      3.2.1. Crear, en caso de no existir, un archivo llamado *cadauno*.
      3.2.2. Cargar en él, el nombre del archivo corriente.
      3.2.3. Cargar cada uno de los elementos contenidos en el multiset propio del archivo, al nuevo.
4. `totales`.
   4.1. Crear, en caso de no existir, un archivo llamado *totales.*
   4.2. Cargar en él cada uno de los elementos contenidos en el multiset de todos los archivos.

Ayuda

En caso de no determinar el path a un directorio, se ofrece al usuario una breve reseña con el propósito del programa.

Procesos implementados

listaordenada

- `lista_t *lista_crear()`. Crea una lista vacía y la devuelve.
  Guarda espacio en memoria, inicializa atributos, retorna la lista.

- `int lista_insertar(lista_t *l, elemento_t elem, unsigned int pos)`. Inserta el elemento `elem` en la posición `pos` de la lista `l`.
  Valida la posición, guarda espacio en memoria para realizar la inserción, busca la posición donde se desea insertar, inserta, incrementa en uno, retorna ejecución exitosa.
- `elemento_t *lista_eliminar(lista_t *l, unsigned int pos)`. Elimina el elemento de la posición `pos` de la lista `l`.
  Valida los parámetros (que no sean nulos y que la posición sea válida), busca la posición anterior a la que se desea eliminar, actualiza referencias, decrementa la cantidad, retorna el elemento contenido en la posición.
- `elemento_t *lista_elemento(lista_t *l, unsigned int pos)`. Devuelve un puntero al elemento que ocupa la posición `pos` de la lista `l`.
  Valida que la posición sea válida, busca la posición donde se desea consultar el elemento, se lo retorna.
- `int lista_ordenar(lista_t *l, funcion_comparar_t comparar)`. Dada la lista `l` y la función `comparar`, ordena la lista de acuerdo con el criterio de dicha función.
  Se utiliza un algoritmo de ordenamiento del tipo mergeSort y, se acuerdo al criterio de la función, se acomodan los elementos. Para tal fin se usa un recorrido 2x1 y así poder hacer la división en mitades de la estructura; por cada elemento que recorre un puntero, recorre dos el otro y, así, se llega a la mitad de la estructura. Se hace moficiación de la lista parametrizada.
- `unsigned int lista_cantidad(lista_t *l)`. Devuelve la cantidad de elementos de la lista `l`.
  Consulta el atributo cantidad de lista y lo retorna.
- `int lista_vacia(lista_t lista)`. Devuelve verdadero (entero diferente de 0) si la lista está vacía, y falso (entero igual que cero) si la lista contiene al menos un elemento.
  Consulta el atributo cantidad de lista, lo evalúa, y retorna el entero en consecuencia.

multiset

- `multiset_t *multiset_crear()`. Crea un multiset vacío de palabras y lo devuelve.
  Guarda espacio en memoria, inicializa los atributos creando y retorna el multiset.
- `void multiset_insertar(multiset_t *m, char *s)`. Inserta la palabra s al multiset m.
  Dada una cadena de caracteres a insertar, recorre uno a uno sus caracteres y se va desplazando por el árbol trie según corresponda (avanzando en los nodos o creando ramas nuevas). Al llegar al nodo correspondiente, actualiza su atributo cantidad.
- `int multiset_cantidad(multiset_t *m, char *s)`. Devuelve la cantidad de repeticiones de la palabra s en el multiset m.
  Dada una cadena de caracteres, recorre uno a uno sus caracteres y se va desplazando por las ramas del árbol trie según corresponda. Retorna el atributo cantidad del nodo correspondiente, o cero, en caso de no encontrar a la cadena en el árbol.
- `lista_t multiset_elementos(multiset_t *m, int (*f)(elemento_t, elemento_t))`. Devuelve una lista de tipo lista t con todos los elementos del multiset m y la cantidad de apariciones de cada uno.
  Se crea una lista donde se almacenarán los pares (cantidad, cadena) y en ella se almacenan las cadenas generadas fruto del recorrido del árbol siempre que corresponda (es decir, el atributo cantidad es diferente de cero).

- `void multiset_eliminar(multiset_t **m)`. Elimina el multiset m liberando el espacio de memoria reservado. Luego de la invocación m debe valer NULL.
  Se libera el espacio en memoria de cada uno de los elementos almacenados en el multiset.
- En cualquier caso, el recorrido del multiset se realiza con un recorrido preorden.

Documentación técnica

No se emplearon herramientas de generación de documentación automática puesto que se realizó la misma durante el desarrollo del programa para la facilitación del entendimiento de código entre las partes de los desarrolladores.

Conclusiones

- Dado que la lista trabaja con posiciones y no se explicita qué número refiere a la primera, se determinó que la primera posición es la número 0.
- Dado que en el prototipo de la función que refiere a la eliminación de elementos de una lista, como el elemento es creado por fuera de listaordenada, se delega la liberación de memoria al archivo que haga uso de la estructura enlazada.
- La experiencia recabada a lo largo del trayecto de aprendizaje universitario estuvo muy presente a lo largo de la realización del proyecto; fue posible evidenciar y tener presente diferentes habilidades, conocimientos y herramientas adquiridos a lo largo de los años previos al día de hoy.