FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS

ASIGNATURA: PROGRAMACIÓN DE COMPUTADORES

PROFESOR: FABIO ANTONIO AVELLANEDA

GESTOR DE ARCHIVOS

AUTORES:

CARLOS ORLNADO BARÓN LEÓN

ANDRÉS FELIPE COCONUBO QUINTERO

Funcionalidad

El programa tiene por objetivo ser un gestor de archivos que permita resolver diferentes problemas que presenta el departamento de finanzas de la empresa Hairy Peter S.A. El programa poseerá la información necesaria sobre todos los archivos que posee la empresa en sus distintos computadores y sucursales. Además de esto, el programa permitirá realizar hojas cálculo sobre la empresa y hacer reportes acerca de estas para los usuarios que los necesiten, después de ellos, estos procesos también se podrán almacenar en los diferentes computadores de la empresa.

Análisis del problema

Para el desarrollo del proyecto se planteó el siguiente esquema con las partes que contiene el código a desarrollar.

1) Sucursal
  1.1) Computadores
    1.1.1) Discos
      1.1.1.1) Documentos
        1.1.1.1.1) Archivos
        1.1.1.1.2) Carpetas
          1.1.1.1.2.1) Archivos
          Tipos de archivos:
          - archivo x
          - hoja de cálculo
          - reporte

El esquema expuesto anteriormente representa las listas que va a poseer el gestor de archivos, estas representan la dirección de memoria en la que se encuentra un archivo en el disco duro. Para desarrollar las listas de direcciones se van a tener cuatro listas de estructuras ordenadas de diferentes maneras, una por tipo, otra por nombre, y otras dos igual que las anteriores pero se manera descendente, con ello el gestor se mantiene organizado todo el tiempo de acuerdo al nombre y extensión de los archivos.
Para la lista de sucursales y computadores se desarrollará otra estructura con la misma funcionalidad pero con diferente nombre, para tener más control sobre las direcciones.

Previsión de riesgos - **Problemas**

Durante el análisis del programa se pudieron encontrar los siguientes problemas con su respectiva solución:

1. Que el documento de texto con la ruta de los archivos esté desorganizado. Para solucionar este problema se crearán primero las listas de las rutas que no estén contenidas en ninguna otra, de esta manera el programa creará las direcciones en orden jerárquico.

2. Que se ingresen datos que no correspondan al tipo que se necesita. Para solucionar este problema se le avisará al usuario por medio de una impresión en pantalla lo que debe ingresar, en caso de lo que lo haga mal y se le pedirá que lo ingrese de nuevo.
3. Que se intenten calcular celdas que no existen. Para solucionar este problema se avisará al usuario por medio de una impresión en pantalla que no se ha podido realizar la operación
4. Que se intenten calcular celdas por medio de una fórmula mal hecha, por ejemplo que posea espacios después de un "+" o posea dos de estos seguidos. Para solucionar este problema, se hará el programa de manera que ignore estos caracteres y calcule la fórmula si el resto de su contenido es consistente.
5. Que se intenten calcular celdas que necesiten de otras antes de calcular las que necesita ésta. Para solucionar este problema se implementará una función que busque las celdas de manera recursiva hasta que encuentre un punto de partida para realizar las fórmulas.

Estructuras

Las estructuras que se usarán durante el desarrollo del programa son las siguientes:

1. Estructura Truta**:** esta estructura se utilizará para hacer las direcciones de discos, directorios y archivos.

```
struct Truta

{

string nombre;

TList <Truta* >  *lista;

char tipo;

};
```

2. Estructura Tnombres: esta estructura se utilizará para guardar los nombres de las carpetas o archivos.

```
struct Tnombres

{

   string izquierda;

   string derecha;

   bool usado=false;

};
```

3. Estructura Thoja: la siguiente estructura representa el formato de una hoja de cálculo sin calcular.

```
struct Thoja

{

int filas, columnas;

string **matriz;

};
```

4. Estructura Tcoordenada: la estructura se utilizará para guardar la coordenada de una celda de acuerdo a letras y números.

```
struct Tcoordenada

{

Char* x;

Char* y;

};
```

5. Estructura Treporte: en esta estructura se guardarán los datos que puede tener un reporte al realizarlo para tener fácil acceso a ellos.

```
struct Treporte

{

string numsem, undprod, undvend, utilioper, utilneta;

};
```

6. Estructuras TAD:

```
template <typename Tdata>

struct TLNode

{

    Tdata info;

    TLNode<Tdata> *next, *prev;

};
```

```
template <typename Tdata>
struct TList
{
    TLNode<Tdata> *first, *last,*window;
};
```

Funciones

1.TAD:

```
template <typename Tdata>
TList<Tdata> * create_list(); Crea la lista.
template <typename Tdata>
void push_front(TList<Tdata> *l, Tdata); Agrega un dato antes del first de la lista.
template <typename Tdata>
void push_back(TList<Tdata> *l, Tdata); Agrega un dato después del last de la lista.
template <typename Tdata>
bool isEmpty(TList<Tdata> *l); Retorna un bool dependiendo si la lista esta vacia o no.
template <typename Tdata>
bool isEnd(TList<Tdata> *l); Retorna un bool dependiendo si la ventana está indefinida o no.
template <typename Tdata>
void gofirst(TList<Tdata> *l); Mueve la ventana al primer elemento de la lista.
template <typename Tdata>
void golast(TList<Tdata> *l); Mueve la ventana al ultimo elemento de la lista.
template <typename Tdata>
void gonext(TList<Tdata> *l); Mueve la ventana al siguiente elemento.
template <typename Tdata>
void goprev(TList<Tdata> *l); Mueve la ventana al anterior elemento.
template <typename Tdata>
```

void insert(TList<Tdata> *l, Tdata info); Agrega un dato antes de la ventana.

template <typename Tdata>

void annex(TList<Tdata> *l, Tdata info); Agrega un dato después de la ventana

template <typename Tdata>

void erase(TList<Tdata> *l); Borra el elemento al que apunta la ventana y la deja en el siguiente.

2. Funciones cadenas

string convertir(string a)	
Parámetro	**String a:** Palabra a la cual se le aplicará la función.
Funcionalidad y relación con otras funciones	Cambia todos los caracteres de la palabra a mayúscula.
Resultado	Devuelve una cadena con las letras en mayúscula.

int tamanocad (string dato)	
Parámetro	**String dato:** En ella se indica el dato que se quiere contar.
Funcionalidad y relación con otras funciones	Permite conocer cuántos caracteres posee una cadena, la función es recursiva.
Resultado	Devuelve el número de caracteres que posee la cadena.

string pegar(string palabra)	
Parámetro	**String palabra:** Palabra a la que se le aplica la función.
Funcionalidad y relación con otras funciones	Si la frase tiene espacios, es decir, posee más de una palabra, quitará los espacios y adicional colocará la extensión “.txt”
Resultado	Retorna la cadena pegada y con la extensión.

int tamanostring(string a)	
Parámetro	**String a:** Palabra a la que se le aplica la función.
Funcionalidad y relación con otras funciones	Cuenta los caracteres de la cadena.
Resultado	Devuelve un entero con la cantidad de caracteres.

char * stoc(string b)	
Parámetro	**String b:** Palabra a la que se le aplica la función.
Funcionalidad y relación con otras funciones	Utiliza la función “tamañosstring”, convierte un string a char*.
Resultado	Retorna un char* con la cadena.

int tamanochar(char* dato)	
Parámetro	**Char*dato:** Dato al que se le aplica la función.
Funcionalidad y relación con otras funciones	Cuenta los caracteres del char*.
Resultado	Retorna un entero con el número de caracteres.

3. Funciones generales del gestor

void agregar(TList<Truta*> *lista,Truta *info)	
Parámetro	**TList <Truta*> *lista:** Este parámetro donde indica la lista donde se encuentra la ruta para imprimir. **Truta *info:** Lista donde se encuentra la información.
Funcionalidad y relación con otras funciones	La función agrega una nueva ruta a la lista. Usa funciones como “convertir”, y algunas del TAD
Resultado	No retorna ningún dato.

void imprimir (TList<Truta*> *l, int espacios)	
Parámetro	**TList <Tdata> *l:** Este parámetro donde indica la lista donde se encuentra la ruta para imprimir. **Int espacios:** Número de espacios

Funcionalidad y relación con otras funciones	Recursivamente va imprimiendo los datos de la lista de carpetas y los archivos que en ella se encuentren.
Resultado	Muestra en pantalla la ruta con los archivos.

TList<Truta*> *create_ruta(TList<Tnombres> *nombres,string elemento)	
Parámetro	**TList <Tnombres> *nombres:** Lista con los nombres de los archivos. **String elemento:** Elemento de comparación.
Funcionalidad y relación con otras funciones	Busca un punto de partida de la lista de forma recursiva, con el cual se creará la carpeta más externa. Usa la función “convertir”.
Resultado	Una vez se hace la conversión de cadena a entero, la función devuelve este dato.

TList<Truta*> * buscar_fichero(TList<string> *direccion,TList<Truta*> *principal)	
Parámetro	**TList <string> *dirección:** Lista con la dirección a buscar. **TList <Truta*>*principal:** Lista en la que se buscará la dirección.
Funcionalidad y relación con otras funciones	Busca el fichero de forma recursiva para utilizarlo después.
Resultado	Devuelve una lista donde está la ruta que se necesita.

void mostrar_fichero(TList<Truta*> *principal)	
Parámetro	**TList <Truta*>*principal:** Ruta que almacena las carpetas más externas y sus subrutas.
Funcionalidad y relación con otras funciones	Utiliza la función “imprimir” para mostrar los archivos que se encuentran.
Resultado	Muestra en pantalla los archivos organizados.

void leer_direccion(TList<string> *direccion)	
Parámetro	**TList <string>*direccion:** Ruta que almacena una ruta.
Funcionalidad y relación con otras funciones	Guarda en una lista cada dato de la ruta, para luego usarla.
Resultado	Inserta en una lista cada dato que conforma la ruta.

void leer_archivo(TList<Tnombres> *nombres, fstream &lectura)	
Parámetro	**TList <Tnombres>*nombres:** En esta lista se guardaran los datos que se encuentran en el archivo de lectura. **Ifstream &lectura:** Archivo que se abrió en la función main con las carpetas y archivos.
Funcionalidad y relación con otras funciones	Guarda en una lista cada dato de la ruta, para luego usarla.
Resultado	No retorna ningún dato, sólo se guarda en una lista los datos leídos.

void crear_nombres(TList<Tnombres> *nombres,TList<Truta*> *principal)	
Parámetro	**TList <Tnombres>*nombres:** En esta lista se guardaran los datos que se encuentran en el archivo de lectura. **Tlist <Truta*>*principal:** Lista donde se encuentran las carpetas más externas.
Funcionalidad y relación con otras funciones	Crea rutas de los nombres según se encuentran. Usa funciones como “convertir”, “agregar”, “créate_ruta”.
Resultado	No retorna ningún dato, sólo se se guarda la información en la lista principal.

int buscar (Thoja hoja,char *dato)	
Parámetro	**Thoja hoja:** Se pasa la hoja que se esté manejando. **Char* dato:** Se busca en dato en en la matriz.

Funcionalidad y relación con otras funciones	Se usa función “atoi” que permite convertir de cadena a entero.
Resultado	Una vez se hace la conversión de cadena a entero, la función devuelve este dato.

int operar_celda (Thoja hojac,char *dato)	
Parámetro	**Thoja hojac:** Con esto sabemos a qué hoja de cálculo se está realizando la operación.
Funcionalidad y relación con otras funciones	De forma recursiva va mirando en cada posición de la matriz, haciendo llamado a la función “buscar” y mientras no sea número, es decir, operación, vuelve a entrar y así mismo va realizando la suma.
Resultado	Una vez se tenga el resultado se asignará a la celda correspondiente.

void calcular_hoja()	
Parámetro	No requiere de de parámetros
Funcionalidad y relación con otras funciones	Abre y crea los archivos de lectura(hoja sin calcular) y escritura(hoja calculada). Llama las funciones de “llenarhoja”, “itoa”, “imprimirhoja”.
Resultado	Guarda la hoja calculada en el archivo de escritura.

TList <string> *reporte()	
Parámetro	No requiere de de parámetros
Funcionalidad y relación con otras funciones	Genera los reportes que se pidan, en los archivos correspondientes.
Resultado	Retorna una lista con los nombres de los archivos creados.

void llenarhoja (Thoja hoja,ifstream &lectura)	
Parámetro	**Thoja hoja:** Hoja a la cual se están llenando los datos. **Ifstream &lectura:** Se pasa el archivo donde se encuentran los datos.
Funcionalidad y relación con otras funciones	Llena una matriz con los datos del archivo.
Resultado	Llena una matriz con los datos que se encuentran en el archivo. No retorna ningún archivo.

void imprimirhoja (Thoja hojac,ofstream &escritura_hoja)	
Parámetro	**Thoja hoja:** Hoja a la cual se están llenando los datos. **ofstream &escritura_hoja:** Se pasa el archivo donde se imprimirá la hoja calculada.
Funcionalidad y relación con otras funciones	Imprime en un archivo la hoja de cálculo, con todas sus celdas calculadas.
Resultado	No retorna ningún valor.

Void descendentelistaextension (TList <Tequipo> * l)	
Parámetro	**TList <Tequipo> * l:** En ella se guarda la información de las sucursales y computadores.
Funcionalidad y relación con otras funciones	Se llama la función insertar o anexar para ordenar descendentemente por extensión.
Resultado	Inserta o anexa el dato donde corresponde de tal manera que quede ordenado.

Void descendentelistanombres (TList <Tequipo> * l)	
Parámetro	**TList <Tequipo> * l :** En ella se guarda la información de las sucursales y computadores.

Funcionalidad y relación con otras funciones	Se llama la función insertar o anexar para ordenar descendentemente por nombre.
Resultado	Inserta o anexa el dato donde corresponde de tal manera que quede ordenado.