

Instituto Tecnológico de Costa Rica.

Ingeniería en Computación.

Estructuras de Datos.

Grupo 02.

Proyecto 1A - Caída de caracteres, Matrix.

Manual técnico.

Estudiantes:

Melany Dahiana Salas Fernández.

Carné: 2021121147.

Moisés Solano Espinoza.

Carné: 2021144322

II Semestre, 2021.

Profesor: Víctor Garro Abarca.

Contenidos.

1.	Estructuras Usadas.....	3
2.	Variables globales.....	3
3.	Funciones.....	4
4.	Programa Principal.	12
5.	Instalación.	17
6.	Pruebas de ejecución.	17

1. Estructuras Usadas.

Estructuras TPista: La estructura contiene un código único para cada pista, una coordenada X y una Y, una pila de 13 caracteres y puntero siguiente que servirá para la creación de la lista enlazada. Además, a un puntero PtrTPista que apunta a elementos de tipo TPista.

```
typedef struct TPista {
    int codigo;
    int X;
    int Y;
    char pila[13];
    TPista* Siguiente;
}*PtrTPista;
```

TPista
int Código
int X
int Y
char pila[13]
Tpista* Siguiente

2. Variables globales.

fuente: Es el tipo de letra que se usa en las pilas de caracteres, es de tipo ALLEGRO_FONT.

anchoPantalla: Variable de tipo int que contiene las dimensiones del ancho de la ventana.

espacioLetras: Esta variable es de tipo int y define que tan separadas están las letras en las pilas de caracteres.

ciclo1: Esta variable es de tipo boolean y se encarga de que el ciclo 'infinito' de la simulación se continúe ejecutando hasta que el usuario indique lo contrario presionando la tecla <<ESC>>.

coorX: Es de tipo int y tiene la coordenada X inicial y con la que se crearan las distintas pistas.

coorY: Es de tipo int y contiene la coordenada Y inicial, siempre se inicializará en 0.

actual: Es una variable de tipo char, será que contenga el primer carácter random de la pila de caracteres de cada pista.

contPistas: Variable de tipo int que lleva la cuenta del total de pistas que se crean de acuerdo con las dimensiones de la pantalla.

contChar: Variable de tipo int que lleva la cuenta del total de caracteres que se despliegan mediante la ejecución del programa.

contAgrupaciones: Variable de tipo int que lleva la cuenta del total de agrupaciones que se hacen durante la ejecución del programa.

timeD: Variable de tipo int que contiene el tiempo que tarda ejecutándose la lluvia de caracteres.

3. Funciones.

- **excepciones.**

Función que revisa que los caracteres que se despliegan en pantalla no sean puntos, paréntesis... Recibe un numero y verifica que no sea el código ascii de alguno de estos tipos de char.

```
bool excepciones(int num) {
    int excepciones[6];
    excepciones[0] = 91;
    excepciones[1] = 92;
    excepciones[2] = 93;
    excepciones[3] = 94;
    excepciones[4] = 95;
    excepciones[5] = 96;

    for (int i = 0; i < 6; i++) {
        if (excepciones[i] == num)
            return false;
    }
    return true;
}
```

- **GenerarRandom.**

Esta función retorna un caracter random, usa la función fastrand.

```
char GenerarRandom()
{
    char c;
    do {
        c = 65 + rand() % (122 - 65);
    } while (excepciones(c) == false);

    return c;
}
```

- **salidaCiclo.**

Esta función recibe una variable de tipo boolean por referencia y un evento de tipo ALLEGRO_EVENT. Compara si el evento recibido es un evento de teclado, si lo es entonces compara si el evento es la tecla <<ESC>>, si lo es cambia el valor de la variable de tipo boolean.

```
void salidaCiclo(bool& ciclo1, ALLEGRO_EVENT& evento) {
    if (evento.type == ALLEGRO_EVENT_KEY_DOWN) {
        if (evento.keyboard.keycode == ALLEGRO_KEY_ESCAPE) {
            ciclo1 = false;
        }
    }
}
```

- **inicializarPistas.**

Recibe un puntero de tipo PtrTPista por referencia y lo pone a apuntar a NULL.

```
void inicializarPistas(PtrTPista& listaP) {
    listaP = NULL;
}
```



- **Crea_pista.**

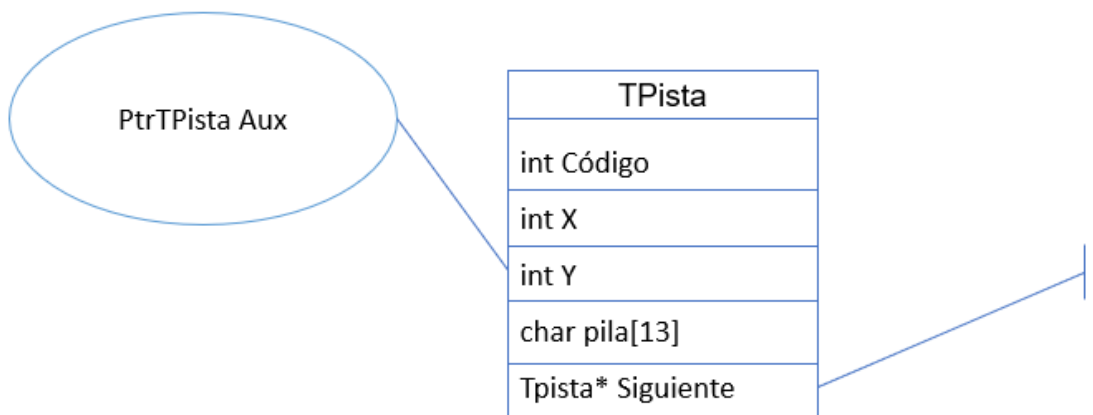
Recibe el código, una coordenada X, una coordenada Y y un carácter. Crea un nuevo puntero de tipo PtrTPista que apunta a un elemento de tipo Tpista y le asigna el código, las coordenadas y la pila, siguiente lo inicializa apuntando a NULL. Retorna el puntero que apunta al nuevo elemento creado.

```
PtrTPista crea_pista(int codigo, int coordX, int coordY, char caracter[1]) {
    PtrTPista Aux = new(Tpista);

    Aux->codigo = codigo;
    Aux->X = coordX;
    Aux->Y = coordY;

    for (int i = 0; i < 13; i++) {
        Aux->pila[i] = caracter[0];
    }

    Aux->Siguiente = NULL;
    return Aux;
}
```

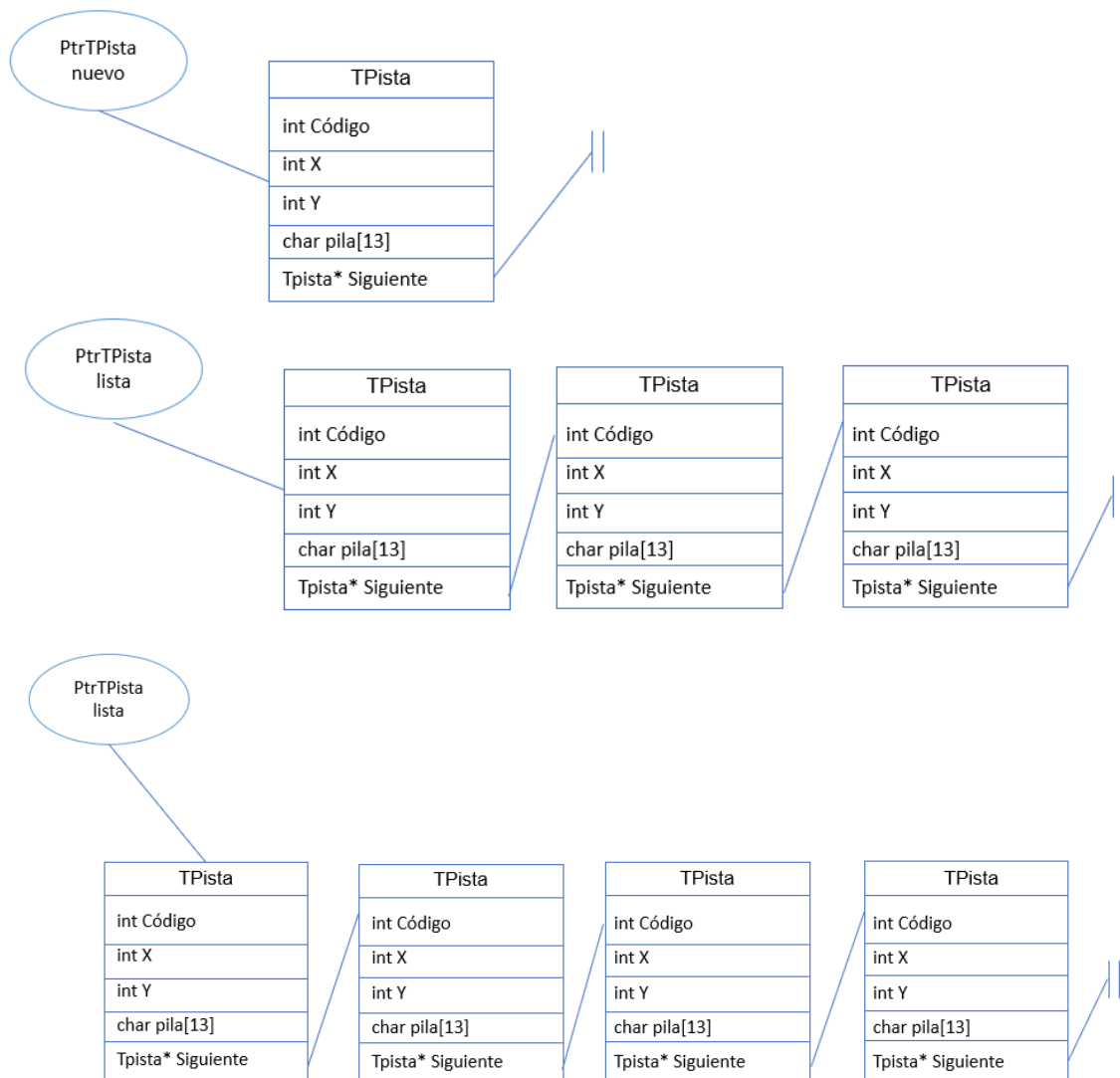


- **agregaLista.**

Agrega elementos a una lista enlazada, recibe la lista por referencia y el nuevo lemento que se va a agregar, ambos punteros de tipo PtrTPista que apuntan a elementos de tipo Tpista.

Agrega al inicio de la lista entonces lo que hace es que siguiente del nuevo elemento para a apuntar al primer elemento de la lista y el puntero que apuntaba a este pasa a apuntar al nuevo elemento.

```
void agregaLista(PtrTPista& lista, PtrTPista nuevo) {
    nuevo->Siguiete = lista; // al siguiente del nuevo elemento
    lista = nuevo; // lista va a ser igual al nuevo que almacena
}
```

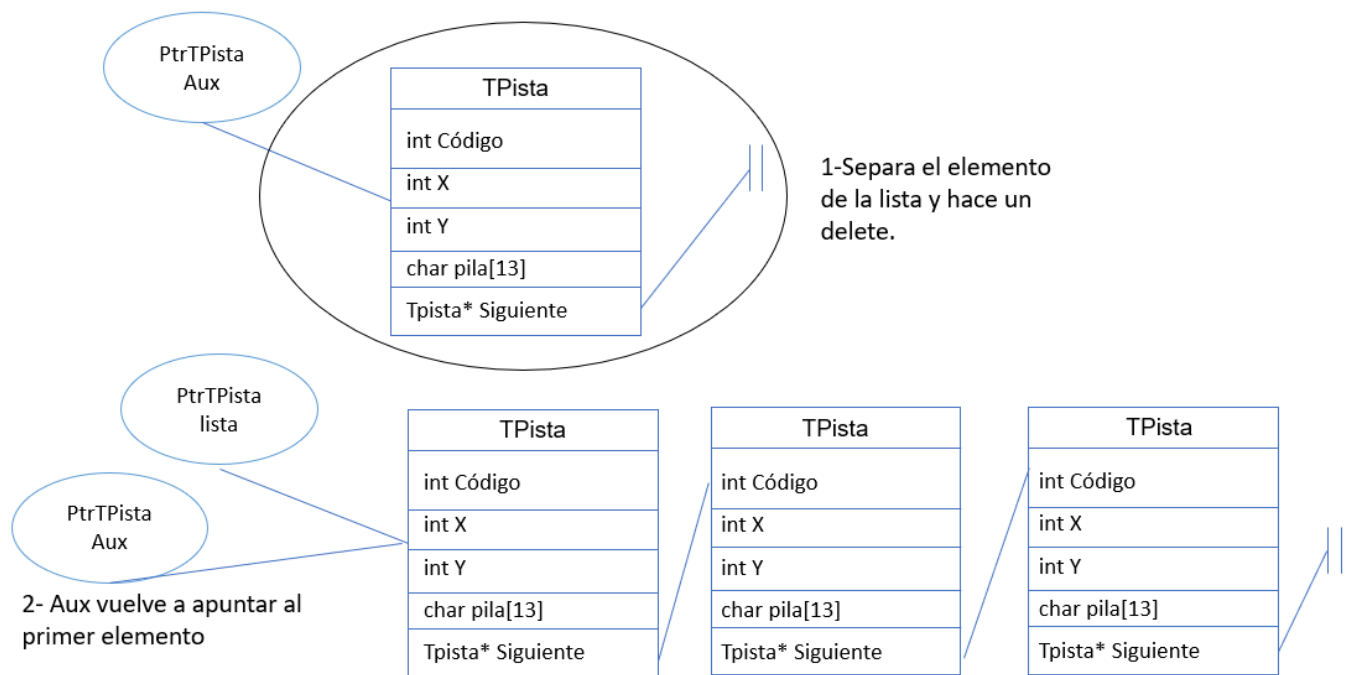


• destruirPistas.

Recibe la lista y crea un puntero auxiliar de tipo PtrTPista, recorre la lista, el puntero auxiliar guarda el valor del primer elemento de la lista y el puntero que apuntaba al primer elemento de la lista pasa a apuntar al segundo, borra el elemento al que apunta el puntero auxiliar, esto lo hace hasta que el primer elemento de la lista se el valor NULL.

```
void destruirPistas(PtrTPista& lista) {
    PtrTPista Aux;
    Aux = lista;
    while (Aux != NULL) {
        lista = lista->Siguiete;
        delete(Aux);
        Aux = lista;
    }

    cout << "La lista ha sido destruida" << endl;
}
```



• BuscarPista.

```
PtrTPista BuscarPista(PtrTPista& Lista, int cual)
{
    bool encontro = false;
    PtrTPista Aux;
    Aux = Lista;
    while (encontro != true && Aux != NULL)
    {
        if (Aux->codigo == cual)
            encontro = true;
        else
            Aux = Aux->Siguiete;
    }
    return Aux;
}
```

Reciba un puntero de tipo PtrTPista y el elemento a buscar mediante su código, esta función usa un ciclo y un puntero auxiliar de tipo PtrTPista para recorrer la lista en busca de un elemento de acuerdo a su código, sale del ciclo cuando encuentra el elemento o cuando el auxiliar apunta a NULL, retorna auxiliar.

- **Borrar1Pista.**

Esta función recibe una lista enlazada por referencia y el código del elemento a borrar. Usa 3 variables auxiliares que son punteros de tipo PtrTPista y una variable booleana, lo primero que hace es llamar a buscarPista para verificar que el elemento a borrar este en la lista enlazada, si elemento no está retorna false. Si la lista tiene 1 solo elemento se le hace un delete a aux que apunta a lista y lista pasa a apuntar a lista-> siguiente que al ser de un elemento tiene como valor NULL.

Si el elemento estaba se entra al ciclo que recorre la lista en busca de este, aquí hay 2 opciones. Si el primer elemento de la lista es el que se quiere borrar, el primer auxiliar está apuntando a este, entonces el puntero que apuntaba al primer elemento pasa a apuntar al siguiente, se hace un delete de aux y la variable booleana para a ser igual a true.

Si el siguiente elemento es el que se va a borrar se usan los otros auxiliares, uno va a tener la dirección de elemento a borrar, el primer auxiliar va a apuntar al elemento antes del que se va a borrar y el tercer auxiliar va a apuntar al elemento que esta después, se hace un delete del auxiliar que tiene el elemento a borrar y el primer auxiliar -> siguiente pasa a apuntar a la dirección de tercer auxiliar, adema se cambia el valor de la variable booleana a true.

```
bool Borrar1Pista(PtrTPista& Lista, int Cual) {
    bool encontro = false; // variable que guarda si se ha encontrado el elemento o no
    PtrTPista Aux, Aux2, AuxBorrar; // punteros para recorrer la lista
    Aux = Lista; // aux va a ser igual a lista
    int contador = 0; // contador para verificar que solo la primera iteración entre al primer if

    // CASO 1: EL ELEMENTO NO SE ENCUENTRA EN LA LISTA ENLAZADA
    if (BuscarPista(Lista, Cual) == NULL) { // si el elemento no está en la lista
        return false;
    }

    // CASO 2: LA LISTA SOLO TIENE UN ELEMENTO Y ES EL QUE SE DESEA ELIMINAR
    if ((Aux != NULL) && (Aux->Siguiente == NULL) && (Aux->codigo == Cual)) { // si la lista tiene solo un elemento
        Lista = Lista->Siguiente; // lista va a ser igual a si misma pero sin el primer elemento
        delete(Aux); // se elimina el primer elemento
        Aux = Lista; // a auxiliar se le asigna nuevamente el valor de lista
        encontro = true;
    }

    // CASO 3: EL ELEMENTO QUE SE QUIERE BORRAR ES EL PRIMERO DE LA LISTA PERO TIENE MAS ELEMENTOS
    while ((Aux != NULL) && !encontro) { // mientras no se haya encontrado y la lista no apunte a NULL
        if ((contador == 0) && Aux->codigo == Cual) { // si el primer elemento es el buscado
            Lista = Lista->Siguiente; // lista va a ser igual a si misma pero sin el primer elemento
            delete(Aux); // se elimina el primer elemento
            Aux = Lista; // a auxiliar se le asigna nuevamente el valor de lista
            contador++;
            encontro = true;
        }

        // CASO 4: EL SIGUIENTE ELEMENTO ES EL QUE SE QUIERE ELIMINAR
        if (Aux->Siguiente != NULL) { // si el siguiente articulo es el que se quiere borrar
            if (Aux->Siguiente->codigo == Cual) {
                AuxBorrar = Aux->Siguiente; // auxBorrar va a apuntar al elemento que se desea borrar
                Aux2 = Aux->Siguiente->Siguiente; // aux2 va a apuntar al siguiente elemento después del borrado
                delete(AuxBorrar); // se borra el elemento
                Aux->Siguiente = Aux2; // se une la parte de antes y después del elemento borrado
                encontro = true;
            }
        }

        Aux = Aux->Siguiente; // avanza de articulo
    }

    return encontro;
}
```


- **moverPista**

Esta función recibe un puntero de tipo PtrTPista que apunta a un elemento que se está desplegando en pantalla, lo que hace es correr los elementos de la pila hacia arriba, es decir, el primero para a ser el segundo, el segundo pasa a ser el tercer y así sucesivamente. Además, aumenta la coordina Y en 20.

```
void moverPista(PtrTPista& actual) {
    actual->pila[12] = actual->pila[11];
    actual->pila[11] = actual->pila[10];
    actual->pila[10] = actual->pila[9];
    actual->pila[9] = actual->pila[8];
    actual->pila[8] = actual->pila[7];
    actual->pila[7] = actual->pila[6];
    actual->pila[6] = actual->pila[5];
    actual->pila[5] = actual->pila[4];
    actual->pila[4] = actual->pila[3];
    actual->pila[3] = actual->pila[2];
    actual->pila[2] = actual->pila[1];
    actual->pila[1] = actual->pila[0];

    actual->Y = actual->Y + 20;
}
```

- **dibujaPantalla.**

Recibe una de las pistas que se están pintando en pantalla y las dibuja en la ventana usando la función de allegro `al_draw_text`, usa la fuente definida en la variables globales e inicializada en el main, para los colores de cambia de acuerdo a la posición que este en la pila de caracteres, donde en carácter de más abajo está en blando y el resto hacen un degradado de verde a un verde muy oscuro para que de la sensación de que los elementos se pierden en la pantalla, además de usa la posición X establecida en el elemento `actual` y a la posición Y se le vas restando de acuerdo los índices de la pila, luego de esto se llama a la función `mover pista` y en la posición 0 se asigna un carácter `random`.

- **PintarTodas**

Se encarga de pintar todas las hileras de caracteres en

```
void dibujaPantalla(PtrTPista& pistaActual, char caracter[1]) {
    if (pistaActual != NULL) {
        actual[0] = pistaActual->pila[0];
        al_draw_text(fuente, al_map_rgb(71, 172, 52), pistaActual->X, pistaActual->Y - 20, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[1];
        al_draw_text(fuente, al_map_rgb(66, 153, 53), pistaActual->X, pistaActual->Y - 40, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[2];
        al_draw_text(fuente, al_map_rgb(63, 140, 53), pistaActual->X, pistaActual->Y - 60, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[3];
        al_draw_text(fuente, al_map_rgb(59, 125, 50), pistaActual->X, pistaActual->Y - 80, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[5];
        al_draw_text(fuente, al_map_rgb(54, 110, 46), pistaActual->X, pistaActual->Y - 100, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[6];
        al_draw_text(fuente, al_map_rgb(50, 94, 41), pistaActual->X, pistaActual->Y - 120, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[7];
        al_draw_text(fuente, al_map_rgb(42, 76, 34), pistaActual->X, pistaActual->Y - 140, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[8];
        al_draw_text(fuente, al_map_rgb(35, 58, 30), pistaActual->X, pistaActual->Y - 160, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[9];
        al_draw_text(fuente, al_map_rgb(27, 43, 25), pistaActual->X, pistaActual->Y - 180, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[10];
        al_draw_text(fuente, al_map_rgb(19, 28, 19), pistaActual->X, pistaActual->Y - 200, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[11];
        al_draw_text(fuente, al_map_rgb(11, 14, 12), pistaActual->X, pistaActual->Y - 220, ALLEGRO_ALIGN_CENTRE, actual);
        actual[0] = pistaActual->pila[12];
        al_draw_text(fuente, al_map_rgb(6, 7, 7), pistaActual->X, pistaActual->Y - 240, ALLEGRO_ALIGN_CENTRE, actual);

        al_draw_text(fuente, al_map_rgb(255, 255, 255), pistaActual->X, pistaActual->Y, ALLEGRO_ALIGN_CENTRE, caracter);
    }

    //al_flip_display();
    moverPista(pistaActual);
    pistaActual->pila[0] = caracter[0];
}
```

pantalla. Recibe la lista enlazada. Se crea un puntero auxiliar para recorrer la lista y una variable tipo char caracter que almacenará una letra aleatoria. En el ciclo while se crea un carácter aleatorio en cada iteración y se aumenta el contador de caracteres mostrados en pantalla, luego se llama a la función de `dibujaPantalla(Aux, caracter)` con el elemento actual de la lista y el caracter. Después el auxiliar avanza de elemento.

```
void pintarTodas(PtrTPista lista) {
    PtrTPista Aux = lista;
    char caracter[1];
    while (Aux != NULL) {
        caracter[0] = GenerarRandom();
        contChar++;
        dibujaPantalla(Aux, caracter);
        Aux = Aux->Siguiete;
    }
}
```

• VerificarExtremo

Recibe la lista por parámetro de referencia ya que se va a editar. Se utiliza para eliminar las hileras de caracteres que desaparecen de la pantalla y evitar que se llene la memoria de basura. Se genera un puntero auxiliar para recorrer la lista enlazada de hileras y mientras no apunte a NULL realiza las validaciones. Si la coordenada Y del elemento actual es mayor o igual al `altoPantalla + 170` se elimina de la lista enlazada porque ya no se ve en la pantalla. En ese momento el auxiliar apunta de nuevo al inicio de la lista enlazada para hacer el resto de las validaciones. Si no se elimina el elemento el auxiliar avanza. La variable de `altoPantalla` es

```
void verificarExtremo(PtrTPista& lista) {
    PtrTPista Aux = lista;
    while (Aux != NULL) {
        if (Aux->Y >= (altoPantalla + 170)) {
            Borrar1Pista(lista, Aux->codigo);
            Aux = lista;
        }
        else
            Aux = Aux->Siguiete;
    }
}
```

global y almacena el tamaño vertical de la pantalla y como se le suma 170 se crea el número máximo en el que puede aparecer una hilera sin ser destruida.

• estaCordenadaDisponible

Esta función es de tipo bool porque retorna la respuesta a si la coordenada de la pista está disponible para imprimir caracteres en ella. Recibe una lista enlazada y un identificador de tipo int para hacer las validaciones que es la posición X. Se crea un puntero auxiliar para recorrer la lista y mientras no apunte a NULL ejecuta el código. Como cada agrupación de caracteres presente en la lista enlazada posee

una coordenada X, se pregunta si es igual que la del buscado y si la coordenada Y del auxiliar es menor a 250. Esto porque el rastro de caracteres de una hilera es un poco menor a 250 entonces se evita que se impriman hileras encima de otras. Si al terminar de recorrer toda la lista enlazada no se cumplen estas condiciones quiere decir que la nueva hilera se puede crear en esa posición X sin ningún inconveniente y retorna true.

```
bool estaCoordenadaDisponible(PtrTPista lista, int cual) {
    PtrTPista Aux = lista;
    while (Aux != NULL) {
        if (cual == Aux->X && Aux->Y < 250)
            return false;
        Aux = Aux->Siguiente;
    }
    return true;
}
```

- **guardarArchivo**

La función de guardar archivo se encarga de guardar los datos de la simulación en un archivo .txt. Los datos almacenados son cantidad total de pistas, cantidad total de caracteres, cantidad total de hileras y los segundos que duró la ejecución. Se crea un puntero FILE y se abre el archivo en formato append, si no existe se crea. Con fprintf_s se van escribiendo los datos en el archivo sin borrar los datos previamente existentes. Al finalizar el guardado se cierra el archivo porque no será utilizado nuevamente.

```
void guardararchivo() {
    FILE* arcEstMatrix;
    fopen_s(&arcEstMatrix, "EstadisticasMatrix.txt", "a");

    fprintf_s(arcEstMatrix, "\n\tESTADISTICAS\t");
    fprintf_s(arcEstMatrix, "\nCANTIDAD TOTAL DE PISTAS: %i", contPistas);
    fprintf_s(arcEstMatrix, "\nCANTIDAD TOTAL DE CARACTERES: %i", contChar);
    fprintf_s(arcEstMatrix, "\nCANTIDAD TOTAL DE HILERAS: %i", contAgrupaciones);
    fprintf_s(arcEstMatrix, "\nCANTIDAD TOTAL DE SEGUNDOS: %i", timeD);
    fprintf_s(arcEstMatrix, "\n-----");

    fclose(arcEstMatrix);
}
```

- **PintarLista**

Esta función recibe la lista enlazada de las agrupaciones de letras y la recorre elemento por elemento. Se crea un puntero auxiliar para recorrer la lista mientras no apunte a NULL. Imprime en pantalla "Elemento número: " + el código del elemento. No se utiliza durante la ejecución del programa final, pero se utilizó para realizar pruebas y verificar que los elementos se borraran exitosamente para evitar el mal manejo de la memoria dinámica.

```

void pintarLista(PtrTPista lista) {
    PtrTPista Aux = lista;

    if (Aux == NULL) {
        cout << "La lista no tiene elementos" << endl;
    }

    while (Aux != NULL) {
        cout << "Elemento numero: " << Aux->codigo << endl;
        Aux = Aux->Siguiete;
    }
}

```

4. Programa Principal.

En el programa principal se crean el puntero lista y nuevo, ambos de tipo PtrTPista y se inicializan llamando a la función inicializarPistas, luego se creas las variables identificador que sirve para identificar elementos, caracter[1] con "", variable de tipo int numY y numX, además de inicializa la variable global coorX con 10.

Se hace un ciclo while que cuenta la cantidad posible de pistas de acuerdo con las dimensiones de la ventana.

Luego de esto se hace el if que verifica que allegro inicie correctamente. Si todo está bien se procede a habilitar los complementos de allegro como el teclado, fuentes, audio...Posteriormente se crea la ventana, fuentes, samples, timers, cola de eventos y se inicializan con los valores respectivos para uno y se agrega cada evento a la cola de eventos. También se inician los 2 timer creados.

```

while (coorX <= anchoPantalla) {    cada
    contPistas++;
    coorX += espacioLetras;
}

```

```

//Habilitacion de complementos
al_install_keyboard(); //teclado
al_init_font_addon(); //fuentes
al_init_ttf_addon(); //fuentes

al_install_audio();
al_init_acodec_addon();

al_reserve_samples(1);
//Elementos a usar
ALLEGRO_DISPLAY* pantalla;
ALLEGRO_KEYBOARD_STATE teclado;
ALLEGRO_FONT* fuenteC = al_load_font("Fonts/CONSOLAB.ttf", 50, NULL);
ALLEGRO_FONT* fuenteD = al_load_font("Fonts/CONSOLAB.ttf", 18, NULL);
fuente = al_load_font("Fonts/matrix code nfi.ttf", 19, NULL);

ALLEGRO_SAMPLE* song;
song = al_load_sample("Musica/matrix.wav");
al_play_sample(song, 0.5, 0.0, 1.0, ALLEGRO_PLAYMODE_LOOP, NULL);

ALLEGRO_EVENT_QUEUE* colaEventos = al_create_event_queue();
ALLEGRO_TIMER* timer = al_create_timer(1.5 / 30.0);
ALLEGRO_TIMER* timer2 = al_create_timer(8.0 / 30.0);

//PANTALLA
pantalla = al_create_display(anchoPantalla, altoPantalla);
al_set_window_position(pantalla, 200, 200);
al_set_window_title(pantalla, "MATRIX");

al_set_new_display_flags(ALLEGRO_WINDOWED);

//Registro de colas de eventos
al_register_event_source(colasEventos, al_get_keyboard_event_source());
al_register_event_source(colasEventos, al_get_timer_event_source(timer));
al_register_event_source(colasEventos, al_get_timer_event_source(timer2));

al_start_timer(timer);
al_start_timer(timer2);

```

Luego de iniciar el ciclo 1 que contiene básicamente la lógica de la lluvia de caracteres, se crean los eventos y se pone la cola a esperar eventos, si los eventos son del timer 1 se limpia la pantalla y entra a un ciclo for, en este ciclo se hace otro ciclo, esta vez un do while que lo que hace es asegurarse que el número en la posible coordenada X sea un múltiplo del espacio entre caracteres definido en las variables globales. Cuando se tiene el número múltiplo de espacioLetras entra a un if que evalúa si esta coordenada está ya ocupada por una pila de caracteres o si hay una pila desplegándose actualmente, si no está ocupada se crea una nueva pista con la función crea_pista enviándole el identificador, coordenadas X y Y y el carácter y esta nueva pista se agrega a la lista enlazada, adicionalmente se le suma uno al identificador (representa al código de cada pista), al contador de agrupaciones y al contador de pistas. Cuando sale del ciclo for se llama a la función pintar todas con la lista enlazada y se dibuja todo en la ventana. En otro bloque a parte del primer if pero aun dentro del primer while

se llama a la función `salidaCiclo` y se le envía el evento y la variable que hace que el ciclo siga, aquí se evalúa si el evento es de teclado y debe salir del ciclo cambiando el valor de la variable.

```
bool ciclo1 = true;
while (ciclo1) {
    ALLEGRO_EVENT evento;
    al_wait_for_event(colaEventos, &evento);
    if (evento.type == ALLEGRO_EVENT_TIMER) {

        if (evento.timer.source == timer) {
            al_clear_to_color(al_map_rgb(0, 0, 0));

            for (int i = 0; i < 3; i++) {
                do {
                    numX = 10 + rand() % (anchoPantalla - 10);
                } while ((numX % espacioLetras != 0));

                if (estaCoordenadaDisponible(lista, numX) == true) {
                    nuevo = crea_pista(identificador, numX, coorY, caracter);
                    agregalista(lista, nuevo);
                    identificador++;
                    cont1++;
                    //numY = 0;
                    numY = -100 + rand() % (0 - -100);
                    coorY = numY;
                    contAgrupaciones++;
                }
            }

            pintarTodas(lista);

            al_flip_display();
        }
        verificarExtremo(lista);
    }
    salidaCiclo(ciclo1, evento);
}
```

Cuando sale de este ciclo se calcula en tiempo que tardo ejecutando la caída de caracteres en la variable `timeD`. Luego se crean las variables que se usaran en la ventada de estadísticas, se unas `ciclo2` como la variable booleana que mantendrá en ciclo hast =a que se indique lo contrario con la tecla <<ESC>>, la variable booleana `color` que es usada para el efecto de parpadeo de la barra baja en la ventana, además se hacen los int to string de los contadores y se guardan en las variables char respectivas. Luego de esto se llama a la función que se encarga de hacer el append al archivo.

```

t1 = clock();
timeD = (double(t1 - t0) / CLOCKS_PER_SEC);
cout << "El tiempo de ejecucion es de: " << timeD << endl;
cout << "El tiempo de ejecucion es de: " << timeD / 60 << endl;

bool ciclo2 = true;
bool color = true;
char bufferAgrupaciones[15];
_itoa_s(contAgrupaciones, bufferAgrupaciones, 10);

char bufferChar[15];
_itoa_s(contChar, bufferChar, 10);

char bufferTime[15];
_itoa_s(timeD, bufferTime, 10);

char seg[] = " segundos";
strcat_s(bufferTime, seg);

char bufferPistas[4];
_itoa_s(contPistas, bufferPistas, 10);

guardararchivo();

```

Posteriormente esta el segundo ciclo, donde encontramos el despliegue de estadísticas en la ventana, aquí nuevamente se crean los eventos y se pone la cola a esperar eventos, se usa el timer2 para que se mantengan las estadísticas en pantalla y se de el efecto de parpadeo de la barra baja, para que todo aparezca en pantalla se usa la función de allego al `_draw_text` y de dibuja todo en pantalla con al `_flip_display`, el `if (color)` que esta en el ciclo sirve para saber si pinta la barra baja en verde o en negro y va cambiando la variable `color` de `true` a `false` y de `false` a `true`.

De igual forma, en el siguiente bloque al primer `if` encontramos la llamada a la función `salidaCiclo` y se le envía el evento y la variable que hace que el ciclo siga, aquí se evalúa si el evento es de teclado y debe salir del ciclo cambiando el valor de la variable.

```

while (ciclo2) {
    ALLEGRO_EVENT evento;
    al_wait_for_event(colaEventos, &evento);
    if (evento.type == ALLEGRO_EVENT_TIMER) {

        if (evento.timer.source == timer2) {

            al_clear_to_color(al_map_rgb(0, 0, 0));
            al_draw_text(fuenteC, al_map_rgb(255, 255, 255), 400, 150, ALLEGRO_ALIGN_LEFT, "E");
            al_draw_text(fuenteC, al_map_rgb(71, 172, 52), 430, 150, ALLEGRO_ALIGN_LEFT, "S");
            al_draw_text(fuenteC, al_map_rgb(66, 153, 53), 460, 150, ALLEGRO_ALIGN_LEFT, "T");
            al_draw_text(fuenteC, al_map_rgb(63, 140, 53), 490, 150, ALLEGRO_ALIGN_LEFT, "A");
            al_draw_text(fuenteC, al_map_rgb(59, 125, 50), 520, 150, ALLEGRO_ALIGN_LEFT, "D");
            al_draw_text(fuenteC, al_map_rgb(54, 110, 46), 550, 150, ALLEGRO_ALIGN_LEFT, "I");
            al_draw_text(fuenteC, al_map_rgb(54, 103, 45), 580, 150, ALLEGRO_ALIGN_LEFT, "S");
            al_draw_text(fuenteC, al_map_rgb(50, 91, 40), 610, 150, ALLEGRO_ALIGN_LEFT, "T");
            al_draw_text(fuenteC, al_map_rgb(43, 76, 35), 640, 150, ALLEGRO_ALIGN_LEFT, "I");
            al_draw_text(fuenteC, al_map_rgb(38, 63, 30), 670, 150, ALLEGRO_ALIGN_LEFT, "C");
            al_draw_text(fuenteC, al_map_rgb(33, 51, 27), 700, 150, ALLEGRO_ALIGN_LEFT, "A");
            al_draw_text(fuenteC, al_map_rgb(29, 43, 25), 730, 150, ALLEGRO_ALIGN_LEFT, "S");

            if (color) {
                al_draw_text(fuenteC, al_map_rgb(255, 255, 255), 760, 150, ALLEGRO_ALIGN_LEFT, "_");
                color = !color;
            }
            else {
                al_draw_text(fuenteC, al_map_rgb(0, 0, 0), 760, 150, ALLEGRO_ALIGN_LEFT, "_");
                color = !color;
            }

            al_draw_text(fuenteD, al_map_rgb(71, 172, 52), 460, 300, ALLEGRO_ALIGN_LEFT, "PISTAS: ");
            al_draw_text(fuenteD, al_map_rgb(255, 255, 255), 540, 300, ALLEGRO_ALIGN_LEFT, bufferPistas);

            al_draw_text(fuenteD, al_map_rgb(71, 172, 52), 460, 340, ALLEGRO_ALIGN_LEFT, "AGRUPACIONES: ");
            al_draw_text(fuenteD, al_map_rgb(255, 255, 255), 600, 340, ALLEGRO_ALIGN_LEFT, bufferAgrupaciones);

            al_draw_text(fuenteD, al_map_rgb(71, 172, 52), 460, 380, ALLEGRO_ALIGN_LEFT, "LETRAS: ");
            al_draw_text(fuenteD, al_map_rgb(255, 255, 255), 540, 380, ALLEGRO_ALIGN_LEFT, bufferChar);

            al_draw_text(fuenteD, al_map_rgb(71, 172, 52), 460, 420, ALLEGRO_ALIGN_LEFT, "TIEMPO: ");
            al_draw_text(fuenteD, al_map_rgb(255, 255, 255), 540, 420, ALLEGRO_ALIGN_LEFT, bufferTime);

            al_flip_display();
        }
    }
    salidaCiclo(ciclo2, evento);
}

```

Por último, encontramos los destroy de los elementos usados durante la ejecución de; programa y de la lista enlazada.

```



al_destroy_event_queue(colaEventos);
al_destroy_font(fuente);
al_destroy_font(fuenteC);
al_destroy_font(fuenteD);
al_destroy_timer(timer);
al_destroy_timer(timer2);
al_destroy_sample(song);
al_destroy_sample_instance(songInstance);
al_destroy_display(pantalla);

destruirPistas(lista);

```


5. Instalación.

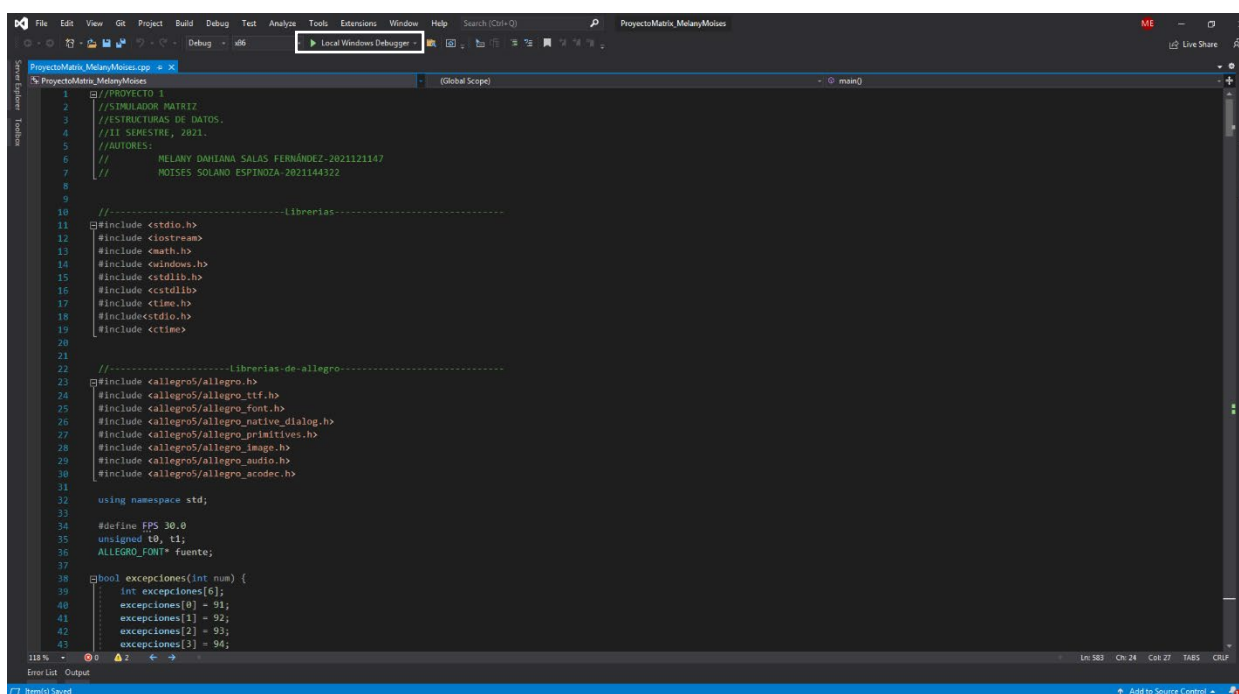
i. Se descarga el .zip con la carpeta del proyecto de Visual Studio.

Nombre	Estado	Fecha de modificación	Tipo	Tamaño
 ProyectoMatrix_MelanyMoises.zip		23/9/2021 18:38	Archivo WinRAR Z...	53 633 KB

ii. Se descomprime el archivo .zip para acceder a los archivos.

iii. Se abre la carpeta del proyecto y se ejecuta el "ProyectoMatrix_MelanyMoises .sln con VisualStudio".

iv. Se ejecuta el proyecto, en el botón verde "Local Windows Debugger".



```

1 //PROYECTO 1
2 //SIMULADOR MATRIX
3 //ESTRUCTURAS DE DATOS.
4 //II SEMESTRE, 2021.
5 //AUTORES:
6 // MELANY DAHIANA SALAS FERNÁNDEZ-2021121147
7 // NOISES SOLANO ESPINOZA-2021144322
8
9
10 //-----librerías-----
11 #include <stdio.h>
12 #include <iostream>
13 #include <math.h>
14 #include <windows.h>
15 #include <stdlib.h>
16 #include <cstdlib>
17 #include <ctime>
18 #include <stdio.h>
19 #include <ctime>
20
21 //-----librerías-de-allegro-----
22 #include <allegro5/allegro.h>
23 #include <allegro5/allegro_ttf.h>
24 #include <allegro5/allegro_font.h>
25 #include <allegro5/allegro_native_dialog.h>
26 #include <allegro5/allegro_primitives.h>
27 #include <allegro5/allegro_image.h>
28 #include <allegro5/allegro_audio.h>
29 #include <allegro5/allegro_acodec.h>
30
31 using namespace std;
32
33 #define FPS 30.0
34 unsigned t0, t1;
35 ALLEGRO_FONT* fuente;
36
37 bool excepciones(int num) {
38     int excepciones[5];
39     excepciones[0] = 01;
40     excepciones[1] = 02;
41     excepciones[2] = 03;
42     excepciones[3] = 04;
43 }

```

v. Ejecución del programa.

6. Pruebas de ejecución.

Mediante este link se puede acceder a un drive con videos y capturas de la ejecución del programa: [https://estudiantec-cr-](https://estudiantec-cr-my.sharepoint.com/:f/g/personal/dhnmelfer_estudiantec_cr/EtNsCjpfqwJKIf1le6wfBAUBwLPVB)

[my.sharepoint.com/:f/g/personal/dhnmelfer_estudiantec_cr/EtNsCjpfqwJKIf1le6wfBAUBwLPVB](https://estudiantec-cr-my.sharepoint.com/:f/g/personal/dhnmelfer_estudiantec_cr/EtNsCjpfqwJKIf1le6wfBAUBwLPVB)
[NPRrw5x0yBrIL9daA?e=FrPGjK](https://estudiantec-cr-my.sharepoint.com/:f/g/personal/dhnmelfer_estudiantec_cr/EtNsCjpfqwJKIf1le6wfBAUBwLPVB)