

## Examen - Programacion avanzada

Alumno: Ángel Alonso Galarza Chávez  
Profesor: Dr. Cuauhtemoc Mancillas López  
Curso: Programación Avanzada

### El juego Yahtzee

Yahtzee es un popular juego de dados basado en la suerte. Se juega con cinco dados y diversas categorías, cada una de las cuales tiene requisitos específicos que deben cumplirse para sumar puntos. El objetivo del juego es obtener la mayor cantidad de puntos posible al lanzar los dados y completar las 13 categorías diferentes.

La parte más esencial del código es la función de simulación. En esta función, se crean listas para las categorías, los dados y los puntajes correspondientes a cada categoría. Cada categoría se selecciona aleatoriamente del arreglo y se elimina para evitar repeticiones. Luego, se inicializan los valores de los dados de manera aleatoria, dentro del rango del 1 al 6. A través de una estructura switch, dependiendo de la categoría seleccionada, se ejecutan diferentes funciones que, según los números obtenidos en los cinco dados, sumarán puntos o no sumarán nada al puntaje. Este puntaje se almacena en el arreglo de puntajes utilizando el índice correspondiente a la categoría.

Finalmente, se suman los puntajes de la lista. Si la suma de los puntajes de las primeras seis categorías es mayor o igual a 63, se añade un bono al puntaje final. Al concluir, se imprimen los dados lanzados para cada categoría, los puntajes obtenidos por categoría, el bono si se ha obtenido, y el puntaje total. Como su nombre indica, esta función simula el juego de Yahtzee y se ejecuta un total de mil veces. En cada iteración, se guardan los resultados de la simulación en un archivo de texto. Al final, se imprimen los valores máximos y mínimos de los puntajes totales obtenidos, así como el puntaje más frecuente

### Código C++

A continuación se presenta el código en C++ del juego de Yahtzee:

```
1 int simulacion() {
2
3     std::ofstream archivo("simulaciones.txt", std::ios::app);
4
5     std::vector<int> numeros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
6     std::vector<int> dados(5);
7     std::vector<int> puntaje(13);
8
9     while (!numeros.empty()) {
10
11         int indiceAleatorio = rand() % numeros.size();
12         int seleccion_categoria = numeros[indiceAleatorio];
13
14         // std::cout << "Categoria seleccionada: " << seleccion_categoria <<
15
16         numeros.erase(numeros.begin() + indiceAleatorio);
17
18         for (int i = 0; i < dados.size(); ++i) {
19             dados[i] = rand() % 6 + 1;
20         }
21
22         switch (seleccion_categoria) {
23             case 1:
24
25                 archivo << "Dados 1: " << " ";
26                 for (int i = 0; i < dados.size(); ++i) {
27                     archivo << dados[i] << " ";
28                 }
29
30                 puntaje[0] = Iguales(dados, 1);
31                 archivo << "\n";
32                 break;
```

```

33 case 2:
34
35     archivo << "Datos 1: " << " ";
36     for (int i = 0; i < datos.size(); ++i) {
37         archivo << datos[i] << " ";
38     }
39
40     puntaje[1] = Iguales(datos, 2);
41     archivo << "\n";
42     break;
43 case 3:
44
45     archivo << "Datos 2: " << " ";
46     for (int i = 0; i < datos.size(); ++i) {
47         archivo << datos[i] << " ";
48     }
49     puntaje[2] = Iguales(datos, 3);
50     archivo << "\n";
51     break;
52 case 4:
53     archivo << "Datos 3: " << " ";
54     for (int i = 0; i < datos.size(); ++i) {
55         archivo << datos[i] << " ";
56     }
57     puntaje[3] = Iguales(datos, 4);
58     archivo << "\n";
59     break;
60 case 5:
61     archivo << "Datos 4: " << " ";
62     for (int i = 0; i < datos.size(); ++i) {
63         archivo << datos[i] << " ";
64     }
65     puntaje[4] = Iguales(datos, 5);
66     archivo << "\n";
67     break;
68 case 6:
69     archivo << "Datos 5: " << " ";
70     for (int i = 0; i < datos.size(); ++i) {
71         archivo << datos[i] << " ";
72     }
73     puntaje[5] = Iguales(datos, 6);
74     archivo << "\n";
75     break;
76 case 7:
77     archivo << "Datos 6: " << " ";
78     for (int i = 0; i < datos.size(); ++i) {
79         archivo << datos[i] << " ";
80     }
81     puntaje[6] = Suerte(datos);
82     archivo << "\n";
83     break;
84 case 8:
85     archivo << "Datos 7: " << " ";
86     for (int i = 0; i < datos.size(); ++i) {
87         archivo << datos[i] << " ";
88     }
89     if (Tipo(datos, 3)) {
90         puntaje[7] = Suerte(datos);
91     }
92     archivo << "\n";
93     break;
94 case 9:
95     archivo << "Datos 8: " << " ";
96     for (int i = 0; i < datos.size(); ++i) {
97         archivo << datos[i] << " ";
98     }
99     if (Tipo(datos, 4)) {
100         puntaje[8] = Suerte(datos);
101     }
102     archivo << "\n";
103     break;
104 case 10:
105     archivo << "Datos 9: " << " ";

```

```

106     for (int i = 0; i < dados.size(); ++i) {
107         archivo << dados[i] << " ";
108     }
109     if (Tipo(dados, 5)) {
110         puntaje[9] = Suerte(dados);
111     }
112     archivo << "\n";
113     break;
114 case 11:
115     archivo << "Datos 10: " << " ";
116     for (int i = 0; i < dados.size(); ++i) {
117         archivo << dados[i] << " ";
118     }
119     if (Secuencia(dados, 0)) {
120         puntaje[10] = 25;
121     }
122     archivo << "\n";
123     break;
124 case 12:
125     archivo << "Datos 11: " << " ";
126     for (int i = 0; i < dados.size(); ++i) {
127         archivo << dados[i] << " ";
128     }
129     if (Secuencia(dados, 1)) {
130         puntaje[11] = 35;
131     }
132     archivo << "\n";
133     break;
134 case 13:
135     archivo << "Datos 12: " << " ";
136     for (int i = 0; i < dados.size(); ++i) {
137         archivo << dados[i] << " ";
138     }
139     archivo << "\n";
140     if (FULL\_HOUSE(dados)) {
141         puntaje[12] = 40;
142     }
143     break;
144 }
145 }
146 int total = 0;
147 int suma = 0;
148 int bono = 0;
149
150 // Sumar los 6 primeros elementos
151 for (int i = 0; i < 6; ++i) {
152     suma += puntaje[i];
153 }
154
155 if (suma >= 63) {
156     bono = 35;
157 }
158
159 // std::cout << "Puntajes de las categorias" << std::endl;
160 for (int i = 0; i < puntaje.size(); ++i) {
161     archivo << puntaje[i] << " ";
162 }
163
164 for (int i = 0; i < 13; ++i) {
165     total += puntaje[i];
166 }
167 total += bono;
168 archivo << bono << " " << total << std::endl;
169
170 archivo.close();
171 return total;
172 }

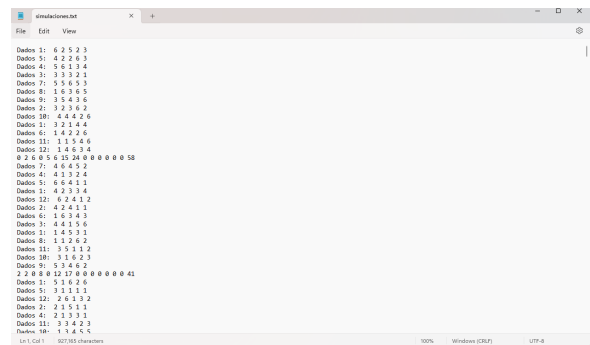
```

## Ejecución

Las siguiente figuras mostraran la ejecución del código y el archivo de texto generado.

```
@Angel on ~/Documents/Cinve
# ./yahtzee.exe
Puntaje minimo: 13
Puntaje maximo: 159
Puntaje mas comun: 32
@Angel on ~/Documents/Cinve
# |
```

(a) Ejecución del juego de Yahtzee



(b) Archivo generado de las simulaciones

Figure 1: Imágenes de la ejecución del programa Yahtzee

## Mensaje Cifrado

Este código en C implementa un cifrado de sustitución simple, en el cual cada letra del alfabeto original es reemplazada por otra letra de manera aleatoria para cifrar un texto.

- **Cifrado:** El proceso de cifrado consiste en sustituir cada letra del texto original por su correspondiente en el alfabeto cifrado. La posición de cada letra en el alfabeto original determina su posición en el alfabeto de sustitución.
- **Descifrado:** Para descifrar el texto, es necesario contar con el alfabeto cifrado utilizado durante el proceso de cifrado. Conociendo este alfabeto, se puede llevar a cabo el proceso inverso para recuperar el texto original.

Una de las funciones esenciales del programa es el desordenamiento del alfabeto, ya que cada letra será sustituida de acuerdo con el nuevo índice del alfabeto mezclado. El programa abre el archivo a cifrar y, leyendo carácter por carácter, realiza el intercambio de letras.

Una vez que se ha generado el archivo cifrado, el programa cierra el archivo y finaliza su ejecución. Para descifrar el archivo, lo primero que se debe hacer es obtener el alfabeto mezclado, que se obtiene leyendo las primeras 27 líneas del archivo cifrado, las cuales corresponden al alfabeto (incluyendo la letra ñ y el espacio). Una vez que se tiene el alfabeto mezclado, se busca el índice del carácter en el mensaje cifrado. Este índice es útil para recuperar el carácter original. Por ejemplo, si la letra 'g' en el alfabeto mezclado tiene un valor ASCII de 103 y se encuentra en la posición 1, esto indica que la letra 'g' reemplazó a la letra 'a'. Con este índice, se puede obtener el carácter 'a' en el alfabeto original.

## Código C

A continuación se muestran los códigos del cifrado del mensaje.

```
1 // Abriendo el archivo txt
2 char filename[] = "mensaje_sin_cifrar.txt";
3 char file_codec[] = "mensaje_cifrado.txt";
4 FILE *archivo = fopen(filename, "r");
5 FILE *archivo_codificado = fopen(file_codec, "w");
6
7 if (archivo == NULL || archivo_codificado == NULL) {
8     printf("Error al abrir uno de los archivo\n");
9 }
10
11 printf("ascii original\n");
12 imprimirAlfabeto(ascii_original);
13
14 printf("Colocando el alfabeto cambiado\n");
15 for (int i = 0; i < 28; i++) {
16     fprintf(archivo_codificado, "%d\n", ascii_alfabeto[i]);
17 }
18
19 printf("Leyendo el archivo de texto original\n");
20 int pass = 0;
```

```

21 char buffer;
22 while ((buffer = fgetc(archivo)) != EOF) {
23     // Condiciones porque se encuentra el caracter especial y no lo puede leer correctamente
24     if (buffer < 0 && pass == 0) {
25         pass = 1;
26     } else if (buffer < 0 && pass == 1) {
27         printf("%c = %c -- %c\n", 164, ascii_alfabeto[27], ascii_original[27]);
28         fputc(ascii_alfabeto[27], archivo_codificado);
29     } else if (buffer != 32) {
30         printf("%c = %c -- %c\n", buffer, ascii_alfabeto[buffer - 96],
31             ascii_original[buffer - 96]);
32         fputc(ascii_alfabeto[buffer - 96], archivo_codificado);
33     } else {
34         fputc(ascii_alfabeto[0], archivo_codificado);
35         printf("%c = %c -- %c\n", buffer, ascii_alfabeto[0], ascii_original[0]);
36     }
37 }
38 fclose(archivo);
39 fclose(archivo_codificado);

```

El siguiente código es el descifrado del mensaje.

```

1  int main() {
2  // Leer del archivo
3  int tamano = 28;
4
5  int ascii_original[28];
6
7  // Inicializar el arreglo con los valores ASCII de 'a' a 'z'
8  for (int i = 0; i < 28; ++i) {
9      if (i == 0) {
10         ascii_original[i] = 32;
11     } else if (i == 27) {
12         ascii_original[i] = 32;
13     } else {
14         ascii_original[i] = 96 + i;
15     }
16 }
17 FILE *archivo = fopen("mensaje_cifrado.txt", "r");
18 if (archivo == NULL) {
19     printf("Error al abrir el archivo.\n");
20     return 1;
21 }
22
23 int alfabeto[tamano];
24 for (int i = 0; i < tamano; i++) {
25     if (fscanf(archivo, "%d", &alfabeto[i]) != 1) {
26         printf("Error al leer el archivo.\n");
27         return 1;
28     }
29 }
30
31 // Obteniendo el alfabeto
32 for (int i = 0; i < tamano; i++) {
33     printf("%d ", alfabeto[i]);
34 }
35 printf("\n");
36
37 printf("Leyendo el archivo de texto codificado\n");
38 char bufferc;
39 int indice = 0;
40 while ((bufferc = fgetc(archivo)) != EOF) {
41     // Condiciones por si encuentra una asignarle su valor en ascii
42     if (bufferc != -16) {
43         if (bufferc == -92) {
44             indice = buscarIndice(alfabeto, 164);
45         } else {
46             indice = buscarIndice(alfabeto, bufferc);
47         }
48         printf("%c", ascii_original[indice]);
49     }
50 }
51
52 fclose(archivo);

```

```

53
54     return 0;
55 }

```

## Ejecución

En las siguientes imágenes se muestran las ejecuciones de los códigos.

```

@Angel on ~ - /Documents/Cinvestav/Cuatrimestre-1/Programacion Avanzada/Programas-C/Examen-programacion 0 main
./mensaje.exe
Alfabeto a partir de los valores ASCII:
32=
97=a
98=b
99=c
100=d
101=
102=f
103=g
104=h
105=i
106=j
107=k
108=l
109=m
110=n
111=i
112=p
113=h
114=r
115=s
116=t
117=u
118=o
119=w
120=x
121=y
122=z
Después del shuffle
Alfabeto a partir de los valores ASCII:
32=
164=n
120=x
121=y
114=r
97=a
117=u
112=p
98=b
99=c
100=d
108=l
118=o
102=f
103=g
109=m
107=k
106=j
111=i
116=t
115=s
113=h
104=
101=
105=w
110=
122=z
164=n

```

(a) Mensaje cifrado

```

@Angel on ~ - /Documents/Cinvestav/Cuatrimestre-1/Programacion Avanzada/Programas-C/Examen-programacion 0 main
./descifrar
32 164 120 121 114 97 117 112 103 98 99 100 108 110 102 101 109 107 106 111 104 105 113 115 119 116 118 122
Leyendo el archivo de texto codificado
alan mathison turing es considerado como uno de los padres de la ciencia de la computacion y precursor de la informatic
a moderna proporciono una formalizacion influyente de los conceptos de algoritmo y computacion la maquina de turing form
ulo su propia version que hoy es ampliamente aceptada como la tesis de churchturing durante la segunda guerra mundial tr
abajo en descifrar los codigos nazis particularmente los de la maquina enigma y durante un tiempo fue el director de la
seccion naval enigma de blatchley park se ha estimado que su trabajo acortó la duracion de esa guerra entre dos y cuatro
a os tras la guerra dise uno de los primeros computadores electronicos programables digitales en el laboratorio
nacional de fisica del reino unido y poco tiempo despues construyo otra de las primeras maquinas en la universidad de ma
nchester en el campo de la inteligencia artificial es conocido sobre todo por la concepcion de la prueba de turing un cr
iterio segun el cual puede juzgarse la inteligencia de una maquina si sus respuestas en la prueba son indistinguibles de
las de un ser humano
@Angel on ~ - /Documents/Cinvestav/Cuatrimestre-1/Programacion Avanzada/Programas-C/Examen-programacion 0 main

```

(b) Mensaje descifrado

Figure 2: Imagenes del cifrado y descifrado del mensaje

## Implementar una lista doblemente ligada

Una lista doblemente enlazada es una estructura de datos lineal en la que cada nodo, además de almacenar un valor, cuenta con dos punteros: uno que señala al siguiente nodo en la lista y otro que apunta al nodo anterior. Esta característica permite el acceso y la modificación de los elementos en ambas direcciones.

La implementación de la lista doblemente enlazada se realiza mediante dos objetos: el objeto nodo y el objeto lista. El objeto nodo contiene el valor del nodo, así como los punteros que enlazan con los nodos adyacentes, es decir, el nodo izquierdo y el nodo derecho.

Por su parte, el objeto lista se encarga de almacenar las referencias a la cabeza y la cola de la lista, además de implementar las funciones típicas asociadas a una lista, como la inserción de nuevos nodos, el ordenamiento y la impresión de la lista

## Código C++

A continuación se mostrara parte del código de la lista doblemente ligada.

```

1 // Creacion de la clase
2 class Nodo {
3 public:
4     int valor;
5     Nodo *izq;
6     Nodo *der;

```

```

7
8  Nodo(int valor) {
9      std::cout << "Creando un nodo con el valor " << valor << std::endl;
10     this->valor = valor;
11     this->izq = nullptr;
12     this->der = nullptr;
13 }
14 Nodo(int valor, Nodo *izquierdo) {
15     std::cout << "Creando un nodo con el valor " << valor
16         << " y asignando un nodo a la izquierda" << std::endl;
17     this->valor = valor;
18     this->izq = izquierdo;
19     this->der = nullptr;
20 }
21 ~Nodo() {
22     std::cout << "Destruyendo un nodo con el valor " << valor
23         << " y volviendo null izquierdo y derecho" << std::endl;
24     this->izq = nullptr;
25     this->der = nullptr;
26 }
27 };
28
29 class Lista {
30 public:
31     Nodo *cabeza;
32     Nodo *cola;
33     Lista() {
34         std::cout << "Creando una lista vacia\n" << std::endl;
35         this->cabeza = nullptr;
36         this->cola = nullptr;
37     }
38     ~Lista() {
39         Nodo *actual = cabeza;
40         while (actual != nullptr) {
41             Nodo *der = actual->der;
42             delete actual;
43             actual = der;
44         }
45     }
46
47     void InsertarValor(int valor) {
48         if (cabeza == nullptr) {
49             std::cout << "Insertando el primer nodo a la lista\n" << std::endl;
50             Nodo *nodo = new Nodo(valor);
51             cabeza = nodo;
52             cola = nodo;
53         } else {
54             std::cout << "Insertando un nodo a la lista\n" << std::endl;
55             Nodo *nodo = new Nodo(valor, cola);
56             cola->der = nodo;
57             cola = nodo;
58         }
59     }
60
61     void InsertarNodoInicio(Nodo *nodo) {
62         if (cabeza == nullptr) {
63             std::cout << "Insertando el primer nodo a la lista\n" << std::endl;
64             cabeza = nodo;
65             cola = nodo;
66         } else {
67             std::cout << "Insertando un nodo al inicio de la lista\n" << std::endl;
68             nodo->der = cabeza;
69             cabeza->izq = nodo;
70             cabeza = nodo;
71         }
72     }
73
74     void InsertarNodoFinal(Nodo *nodo) {
75         if (cabeza == nullptr) {
76             std::cout << "Insertando el primer nodo a la lista\n" << std::endl;
77             cabeza = nodo;
78             cola = nodo;
79         } else {

```

```

80     std::cout << "Insertando un nodo al final de la lista\n" << std::endl;
81     cola->der = nodo;
82     nodo->izq = cola;
83     cola = nodo;
84 }
85 }
86
87 void InsertarNodoPosicion(Nodo *nodo, int posicion) {
88     if (posicion <= 1) {
89         std::cout << "Se insertara al inicio el nuevo nodo con valor "
90             << nodo->valor << std::endl;
91         InsertarNodoInicio(nodo);
92         return;
93     }
94
95     Nodo *actual = cabeza;
96     while (actual != nullptr && 0 < posicion) {
97         actual = actual->der;
98         posicion--;
99     }
100
101     if (actual == nullptr) {
102         std::cout << "Posicion fuera de rango. Se Insertara al final de la lista "
103             << "el nodo con valor "
104             << nodo->valor << std::endl;
105     }
106
107     // Enlazar los nodos
108     nodo->der = actual;
109     nodo->izq = actual->izq;
110     actual->izq->der = nodo;
111     actual->izq = nodo;
112 }
113 void EliminarNodo(Nodo *nodo) {
114     std::cout << "Destruyendo el nodo con el valor " << nodo->valor
115         << std::endl;
116     if (nodo == cabeza) {
117         cabeza = cabeza->der;
118         if (cabeza != nullptr) {
119             cabeza->izq = nullptr;
120         } else {
121             cola = nullptr;
122         }
123     }
124
125     else if (nodo == cola) {
126         cola = cola->izq;
127         if (cola != nullptr) {
128             cabeza->der = nullptr;
129         } else {
130             cabeza = nullptr;
131         }
132     } else {
133         nodo->izq->der = nodo->der;
134         nodo->der->izq = nodo->izq;
135     }
136
137     delete nodo;
138 }
139
140 void Ordenamiento() {
141     std::cout << "Ordenando la lista" << std::endl;
142     Nodo *actual = cabeza->der;
143     while (actual != nullptr) {
144         int temp = actual->valor;
145         Nodo *anterior = actual->izq;
146
147         while (anterior != nullptr && temp < anterior->valor) {
148             actual->valor = anterior->valor;
149             actual = anterior;
150             anterior = anterior->izq;
151         }
152

```



```

153     actual->valor = temp;
154     actual = actual->der;
155 }
156 }
157
158 static void RecorrerAdelante(Nodo *nodo) {
159     if (nodo == nullptr) {
160         std::cout << "\n";
161         return;
162     }
163     std::cout << nodo->valor << " ";
164     RecorrerAdelante(nodo->der);
165 }
166
167 void RecorrerAtras() {
168     Nodo *actual = this->cola;
169     while (actual != nullptr) {
170         std::cout << actual->valor << " \n" << std::endl;
171         actual = actual->izq;
172     }
173 }
174
175 // Sobrecarga del operador + para concatenar listas
176 Lista operator+(const Lista &otraLista) const {
177     Lista nuevaLista;
178
179     // Copiar los elementos de la lista actual a la nueva lista
180     Nodo *actual = cabeza;
181     while (actual != nullptr) {
182         nuevaLista.InsertarNodoFinal(actual); // Pasamos el nodo directamente
183         actual = actual->der;
184     }
185
186     // Copiar los elementos de la otra lista a la nueva lista
187     actual = otraLista.cabeza;
188     while (actual != nullptr) {
189         nuevaLista.InsertarNodoFinal(actual);
190         actual = actual->der;
191     }
192
193     return nuevaLista;
194 }
195
196 friend std::ostream &operator<<(std::ostream &os, const Lista &lista) {
197     RecorrerAdelante(lista.cabeza);
198     return os;
199 }
200 };

```

## Ejecución

A continuación se mostraran las imágenes de la lista doblemente ligada con las funciones que se pueden realizar.

```

@Angel on ~ - Documents/Cinvestav/Cuatrimestre-1/Programacion Avanzada/Programas-C/Examen-programacion C/main
/Lista
Creando una lista vacia
Creando un nodo con el valor 2
Creando un nodo con el valor 3
Creando un nodo con el valor 15
Creando un nodo con el valor 5
Creando un nodo con el valor 1
Insertando el primer nodo a la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
2 3 15 5 1
Destruyendo el nodo con el valor 3
Destruyendo un nodo con el valor 3 y volviendo null izquierdo y derecho
2 15 5 1
Creando un nodo con el valor 17
Insertando un nodo al inicio de la lista
Creando un nodo con el valor 18
17 2 15 5 18 1
Ordenando la lista
1 2 5 10 15 17
Creando una lista vacia
Creando un nodo con el valor 9
Creando un nodo con el valor 33
Creando un nodo con el valor 25
Creando un nodo con el valor 14
Creando un nodo con el valor 3
Insertando el primer nodo a la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
9 33 25 14 3
Concatenando dos listas

```

(a) Primera parte de la ejecución de la lista

```

Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
9 33 25 14 3
Concatenando dos listas
Creando una lista vacia
Insertando el primer nodo a la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Insertando un nodo al final de la lista
Ordenando la nueva lista
Ordenando la lista
1 2 3 5 9 10 14 15 17 25 33
Destruyendo un nodo con el valor 1 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 2 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 3 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 5 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 9 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 10 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 14 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 15 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 17 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 25 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 33 y volviendo null izquierdo y derecho
Destruyendo un nodo con el valor 14 y volviendo null izquierdo y derecho
@Angel on ~ - Documents/Cinvestav/Cuatrimestre-1/Programacion Avanzada/Programas-C/Examen-programacion C/main
/

```

(b) Segunda parte de la ejecución de la lista

Figure 3: Imágenes de la ejecución del programa de la lista doblemente ligada

## Árbol n-ario

Un árbol N-ario es una estructura de datos jerárquica en la que cada nodo puede tener un número variable de hijos, hasta un máximo especificado (N). A diferencia de un árbol binario, donde cada nodo puede tener como máximo dos hijos (izquierdo y derecho), en un árbol N-ario no existe un límite fijo en la cantidad de hijos, aunque se establece un máximo que es N.

El código implementa una clase `Nodo` para representar cada nodo del árbol N-ario y una clase `Arbol` para representar la estructura del árbol en sí. En esta implementación, el número máximo de hijos que puede tener cada nodo se ha fijado en tres. Si se supera este límite, los nodos adicionales se eliminan, garantizando así que el árbol N-ario se mantenga equilibrado.

El código proporcionado ofrece una implementación básica y funcional de un árbol N-ario en C++. Al comprender los conceptos fundamentales y las características de esta estructura de datos, podrás aplicarla en una variedad de problemas de programación.

## Código C++

A continuación se presentan las clases para la construcción del árbol n-ario como la clase `nodo` y la clase `arbol`.

```

1  class Nodo {
2  public:
3      int valor;
4      Nodo **hijos;
5      int numHijos;
6      int capacidad;
7
8      Nodo(int val, int capacidad) : valor(val), capacidad(capacidad), numHijos(0) {
9          hijos = new Nodo *[capacidad];
10     }
11
12     ~Nodo() {
13         std::cout << "Eliminando los hijos del nodo " << this->valor << std::endl;
14         delete[] hijos;
15     }
16
17     // Metodo para agregar un hijo
18     bool agregarHijo(Nodo *hijo) {
19         if (numHijos < capacidad) {
20             hijos[numHijos++] = hijo;
21             return true;
22         }
23         std::cout << "Supero la capacidad de agregar mas hijos" << std::endl;
24         return false;
25     }

```

```

26
27 bool InsertarHijos() {
28     for (int i = 0; i < 3; ++i) {
29         int valor = 1 + rand() % 15;
30         Nodo *nuevoHijo = new Nodo(valor, capacidad);
31         if (!agregarHijo(nuevoHijo)) {
32             // Si no se pudo agregar un hijo, se eliminan los hijos ya agregados
33             for (int j = 0; j < i; ++j) {
34                 delete hijos[j];
35             }
36             return false;
37         }
38     }
39     return true;
40 }
41 };
42
43 class Arbol {
44 public:
45     Nodo *raiz;
46     int capacidadHijos;
47
48     Arbol(int valorRaiz, int capacidadHijos) {
49         raiz = new Nodo(valorRaiz, capacidadHijos);
50         this->capacidadHijos = capacidadHijos;
51     }
52
53     // Destructor
54     ~Arbol() {
55         std::cout << "Eliminado arbol" << std::endl;
56         eliminarNodo(raiz);
57     }
58
59     // Metodo recursivo para eliminar nodos
60     void eliminarNodo(Nodo *nodo) {
61         if (nodo) {
62             for (int i = 0; i < nodo->numHijos; i++) {
63                 eliminarNodo(nodo->hijos[i]);
64             }
65             delete nodo;
66         }
67     }
68
69     // Metodo para agregar tres hijos a un nodo especifico
70     void InsertarNodos(Nodo *nodo) {
71         if (nodo) {
72             nodo->InsertarHijos();
73         } else {
74             std::cout << "Nodo no encontrado." << std::endl;
75         }
76     }
77
78     // Metodo recursivo para imprimir el arbol
79     void imprimirArbol(Nodo *nodo, int nivel = 0) {
80         if (nodo) {
81             for (int i = 0; i < nivel; i++) {
82                 std::cout << "  ";
83             }
84             std::cout << nodo->valor << std::endl;
85             for (int i = 0; i < nodo->numHijos; i++) {
86                 imprimirArbol(nodo->hijos[i], nivel + 1);
87             }
88         }
89     }
90
91     void imprimir() { imprimirArbol(raiz); }
92 };

```

## Ejecución

Las siguiente imagen muestran la ejecución del árbol n-ario.

```

@Angel on ~/Documents/Cinvestav/Cuatrimestre-1/Programacion Avanzada/Programas-C/Examen-programacion
# ./arbol
1 void eliminarNodo(nodo *nodo) {
9     if (!nodo) {
13         for (int i = 0; i < nodo->numHijos; i++) {
8             eliminarNodo(nodo->hijos[i]);
8         }
5         delete nodo;
11     }
7     }
15
8 void insertarNodos(nodo *nodo) {
6     if (!nodo) {
8         nodo = insertarHijos();
8     }
Eliminado arbol:
Eliminando los hijos del nodo 13 encontrado "a" << std::endl;
Eliminando los hijos del nodo 8
Eliminando los hijos del nodo 8
Eliminando los hijos del nodo 9
Eliminando los hijos del nodo 11
Eliminando los hijos del nodo 7 "nodo" int nivel = 0) {
Eliminando los hijos del nodo 15
Eliminando los hijos del nodo 5 nivel = nivel + 1) {
Eliminando los hijos del nodo 6
Eliminando los hijos del nodo 8
Eliminando los hijos del nodo 8
Eliminando los hijos del nodo 8 "nodo" int nivel = 0) {
Eliminando los hijos del nodo 1

```

Figure 4: Primera parte de la ejecución de la lista