Examen - Programacion avanzada

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

El juego Yahtzee

ahtzee 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:

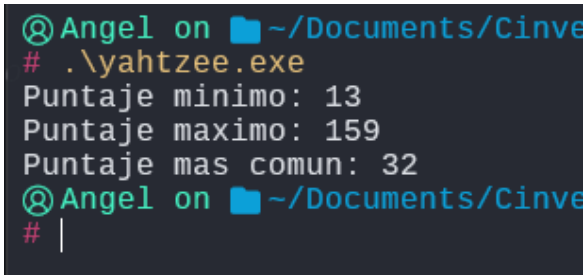
```
int simulacion() {

  std::ofstream archivo("simulaciones.txt", std::ios::app);

  std::vector<int> numeros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
  std::vector<int> dados(5);
  std::vector<int> puntaje(13);

  while (!numeros.empty()) {

    int indiceAleatorio = rand() % numeros.size();
    int seleccion_categoria = numeros[indiceAleatorio];

    // std::cout << "Categoria seleccionada: " << seleccion_categoria <<

    numeros.erase(numeros.begin() + indiceAleatorio);

    for (int i = 0; i < dados.size(); ++i) {
      dados[i] = rand() % 6 + 1;
    }

    switch (seleccion_categoria) {
    case 1:

      archivo << "Dados 1: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }

      puntaje[0] = Iguales(dados, 1);
      archivo << "\n";
      break;
```

```
    case 2:

      archivo << "Dados 1: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }

      puntaje[1] = Iguales(dados, 2);
      archivo << "\n";
      break;
    case 3:

      archivo << "Dados 2: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      puntaje[2] = Iguales(dados, 3);
      archivo << "\n";
      break;
    case 4:
      archivo << "Dados 3: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      puntaje[3] = Iguales(dados, 4);
      archivo << "\n";
      break;
    case 5:
      archivo << "Dados 4: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      puntaje[4] = Iguales(dados, 5);
      archivo << "\n";
      break;
    case 6:
      archivo << "Dados 5: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      puntaje[5] = Iguales(dados, 6);
      archivo << "\n";
      break;
    case 7:
      archivo << "Dados 6: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      puntaje[6] = Suerte(dados);
      archivo << "\n";
      break;
    case 8:
      archivo << "Dados 7: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      if (Tipo(dados, 3)) {
        puntaje[7] = Suerte(dados);
      }
      archivo << "\n";
      break;
    case 9:
      archivo << "Dados 8: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      if (Tipo(dados, 4)) {
        puntaje[8] = Suerte(dados);
      }
      archivo << "\n";
      break;
    case 10:
      archivo << "Dados 9: " << " ";
```

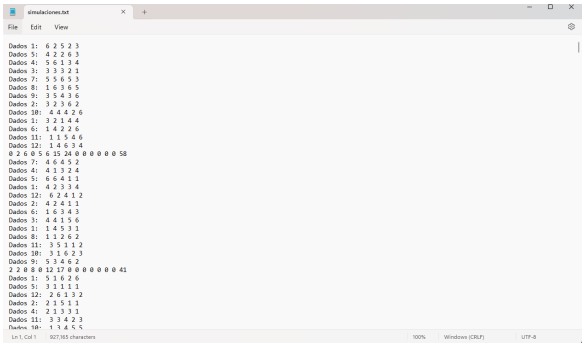
```
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      if (Tipo(dados, 5)) {
        puntaje[9] = Suerte(dados);
      }
      archivo << "\n";
      break;
    case 11:
      archivo << "Dados 10: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      if (Secuencia(dados, 0)) {
        puntaje[10] = 25;
      }
      archivo << "\n";
      break;
    case 12:
      archivo << "Dados 11: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      if (Secuencia(dados, 1)) {
        puntaje[11] = 35;
      }
      archivo << "\n";
      break;
    case 13:
      archivo << "Dados 12: " << " ";
      for (int i = 0; i < dados.size(); ++i) {
        archivo << dados[i] << " ";
      }
      archivo << "\n";
      if (FULL\_HOUSE(dados)) {
        puntaje[12] = 40;
      }
      break;
    }
  }
  int total = 0;
  int suma = 0;
  int bono = 0;

  // Sumar los 6 primeros elementos
  for (int i = 0; i < 6; ++i) {
    suma += puntaje[i];
  }

  if (suma >= 63) {
    bono = 35;
  }

  // std::cout << "Puntajes de las categorias" << std::endl;
  for (int i = 0; i < puntaje.size(); ++i) {
    archivo << puntaje[i] << " ";
  }

  for (int i = 0; i < 13; ++i) {
    total += puntaje[i];
  }
  total += bono;
  archivo << bono << " " << total << std::endl;

  archivo.close();
  return total;
}
```

Ejecución

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



(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.

```
    // Abriendo el archivo txt
char filename[] = "mensaje_sin_cifrar.txt";
char file_codec[] = "mensaje_cifrado.txt";
FILE *archivo = fopen(filename, "r");
FILE *archivo_codificado = fopen(file_codec, "w");

if (archivo == NULL || archivo_codificado == NULL) {
  printf("Error al abrir uno de los archivo\n");
}

printf("ascii original\n");
imprimirAlfabeto(ascii_original);

printf("Colocando el alfabeto cambiado\n");
for (int i = 0; i < 28; i++) {
  fprintf(archivo_codificado, "%d\n", ascii_alfabeto[i]);
}

printf("Leyendo el archivo de texto original\n");
int pass = 0;
```

```
char buffer;
while ((buffer = fgetc(archivo)) != EOF) {
  // Condiciones porque se encuentra el caracter especial y no lo puede leer correctamente
  if (buffer < 0 && pass == 0) {
    pass = 1;
  } else if (buffer < 0 && pass == 1) {
    printf("%c = %c -- %c\n", 164, ascii_alfabeto[27], ascii_original[27]);
    fputc(ascii_alfabeto[27], archivo_codificado);
  } else if (buffer != 32) {
    printf("%c = %c -- %c\n", buffer, ascii_alfabeto[buffer - 96],
           ascii_original[buffer - 96]);
    fputc(ascii_alfabeto[buffer - 96], archivo_codificado);
  } else {
    fputc(ascii_alfabeto[0], archivo_codificado);
    printf("%c = %c -- %c\n", buffer, ascii_alfabeto[0], ascii_original[0]);
  }
}
fclose(archivo);
fclose(archivo_codificado);
```

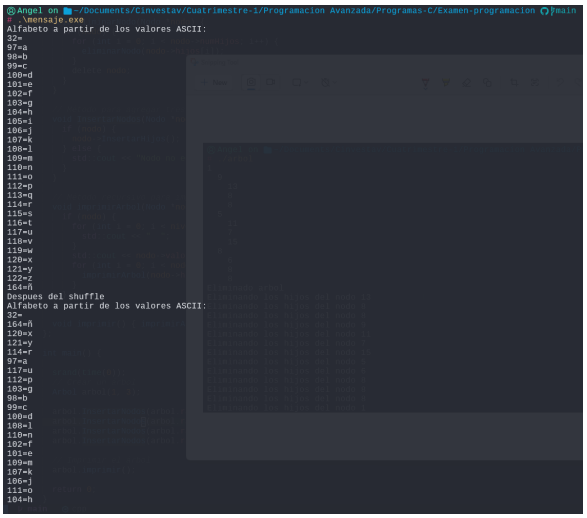
El siguiente codigo es el descifrado del mensaje.

```
  int main() {
// Leer del archivo
int tamano = 28;

int ascii_original[28];

// Inicializar el arreglo con los valores ASCII de 'a' a 'z'
for (int i = 0; i < 28; ++i) {
  if (i == 0) {
    ascii_original[i] = 32;
  } else if (i == 27) {
    ascii_original[i] = 32;
  } else {
    ascii_original[i] = 96 + i;
  }
}
FILE *archivo = fopen("mensaje_cifrado.txt", "r");
if (archivo == NULL) {
  printf("Error al abrir el archivo.\n");
  return 1;
}

int alfabeto[tamano];
for (int i = 0; i < tamano; i++) {
  if (fscanf(archivo, "%d", &alfabeto[i]) != 1) {
    printf("Error al leer el archivo.\n");
    return 1;
  }
}

// Opteniendo el alfabeto
for (int i = 0; i < tamano; i++) {
  printf("%d ", alfabeto[i]);
}
printf("\n");

printf("Leyendo el archivo de texto codificado\n");
char bufferc;
int indice = 0;
while ((bufferc = fgetc(archivo)) != EOF) {
  // Condiciones por si encuentra una    asignarle su valor en ascii
  if (bufferc != -16) {
    if (bufferc == -92) {
      indice = buscarIndice(alfabeto, 164);
    } else {
      indice = buscarIndice(alfabeto, bufferc);
    }
    printf("%c", ascii_original[indice]);
  }
}

fclose(archivo);
```

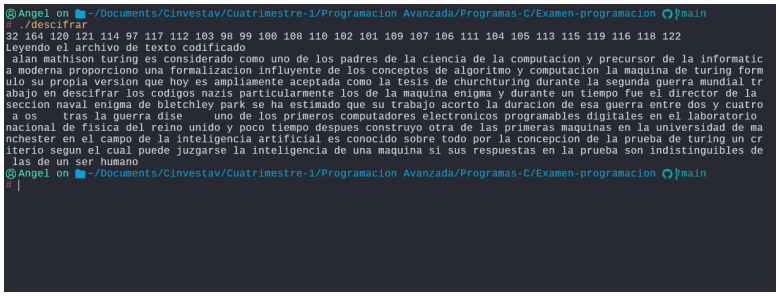
```
53
54    return 0;
55  }
```

Ejecución

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



(a) Mensaje cifrado



(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;
```

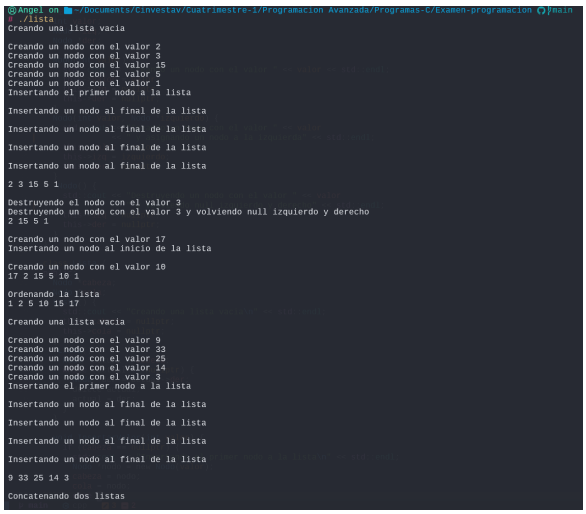
```
7
8   Nodo(int valor) {
9     std::cout << "Creando un nodo con el valor " << valor << std::endl;
10    this->valor = valor;
11    this->izq = nullptr;
12    this->der = nullptr;
13  }
14  Nodo(int valor, Nodo *izquierdo) {
15    std::cout << "Creando un nodo con el valor " << valor
16              << " 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 {
```

```
    std::cout << "Insertando un nodo al final de la lista\n" << std::endl;
    cola->der = nodo;
    nodo->izq = cola;
    cola = nodo;
  }
}

void InsertarNodoPosicion(Nodo *nodo, int posicion) {
  if (posicion <= 1) {
    std::cout << "Se insertara al inicio el nuevo nodo con valor "
              << nodo->valor << std::endl;
    InsertarNodoInicio(nodo);
    return;
  }

  Nodo *actual = cabeza;
  while (actual != nullptr && 0 < posicion) {
    actual = actual->der;
    posicion--;
  }

  if (actual == nullptr) {
    std::cout << "Posicion fuera de rango. Se Insertara al final de la lista "
                 "el nodo con valor "
              << nodo->valor << std::endl;
  }

  // Enlazar los nodos
  nodo->der = actual;
  nodo->izq = actual->izq;
  actual->izq->der = nodo;
  actual->izq = nodo;
}
void EliminarNodo(Nodo *nodo) {
  std::cout << "Destruyendo el nodo con el valor " << nodo->valor
            << std::endl;
  if (nodo == cabeza) {
    cabeza = cabeza->der;
    if (cabeza != nullptr) {
      cabeza->izq = nullptr;
    } else {
      cola = nullptr;
    }
  }

  else if (nodo == cola) {
    cola = cola->izq;
    if (cola != nullptr) {
      cabeza->der = nullptr;
    } else {
      cabeza = nullptr;
    }
  } else {
    nodo->izq->der = nodo->der;
    nodo->der->izq = nodo->izq;
  }

  delete nodo;
}

void Ordenamiento() {
  std::cout << "Ordenando la lista" << std::endl;
  Nodo *actual = cabeza->der;
  while (actual != nullptr) {
    int temp = actual->valor;
    Nodo *anterior = actual->izq;

    while (anterior != nullptr && temp < anterior->valor) {
      actual->valor = anterior->valor;
      actual = anterior;
      anterior = anterior->izq;
    }

```

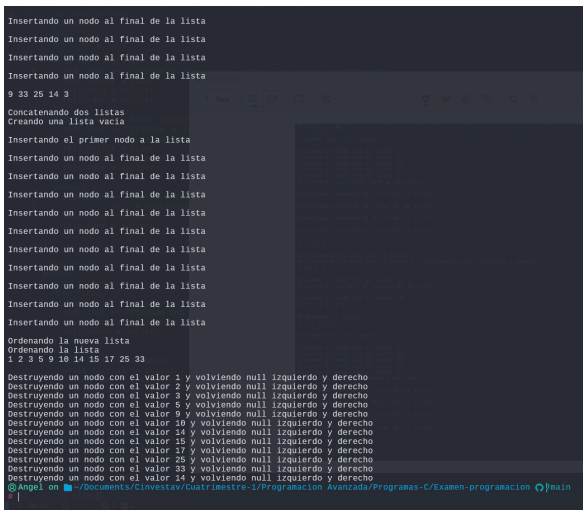
```
      actual->valor = temp;
      actual = actual->der;
    }
  }

  static void RecorrerAdelante(Nodo *nodo) {
    if (nodo == nullptr) {
      std::cout << "\n";
      return;
    }
    std::cout << nodo->valor << " ";
    RecorrerAdelante(nodo->der);
  }

  void RecorrerAtras() {
    Nodo *actual = this->cola;
    while (actual != nullptr) {
      std::cout << actual->valor << " \n" << std::endl;
      actual = actual->izq;
    }
  }

  // Sobrecarga del operador + para concatenar listas
  Lista operator+(const Lista &otraLista) const {
    Lista nuevaLista;

    // Copiar los elementos de la lista actual a la nueva lista
    Nodo *actual = cabeza;
    while (actual != nullptr) {
      nuevaLista.InsertarNodoFinal(actual); // Pasamos el nodo directamente
      actual = actual->der;
    }

    // Copiar los elementos de la otra lista a la nueva lista
    actual = otraLista.cabeza;
    while (actual != nullptr) {
      nuevaLista.InsertarNodoFinal(actual);
      actual = actual->der;
    }

    return nuevaLista;
  }

  friend std::ostream &operator<<(std::ostream &os, const Lista &lista) {
    RecorrerAdelante(lista.cabeza);
    return os;
  }
};
```

Ejecución

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



(a) Primera parte de la ejecución de la lista



(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.

```
    class Nodo {
public:
  int valor;
  Nodo **hijos;
  int numHijos;
  int capacidad;

  Nodo(int val, int capacidad) : valor(val), capacidad(capacidad), numHijos(0) {
    hijos = new Nodo *[capacidad];
  }

  ~Nodo() {
    std::cout << "Eliminando los hijos del nodo " << this->valor << std::endl;
    delete[] hijos;
  }

  // Metodo para agregar un hijo
  bool agregarHijo(Nodo *hijo) {
    if (numHijos < capacidad) {
      hijos[numHijos++] = hijo;
      return true;
    }
    std::cout << "Supero la capacidad de agregar mas hijos" << std::endl;
    return false;
  }
```

```
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.

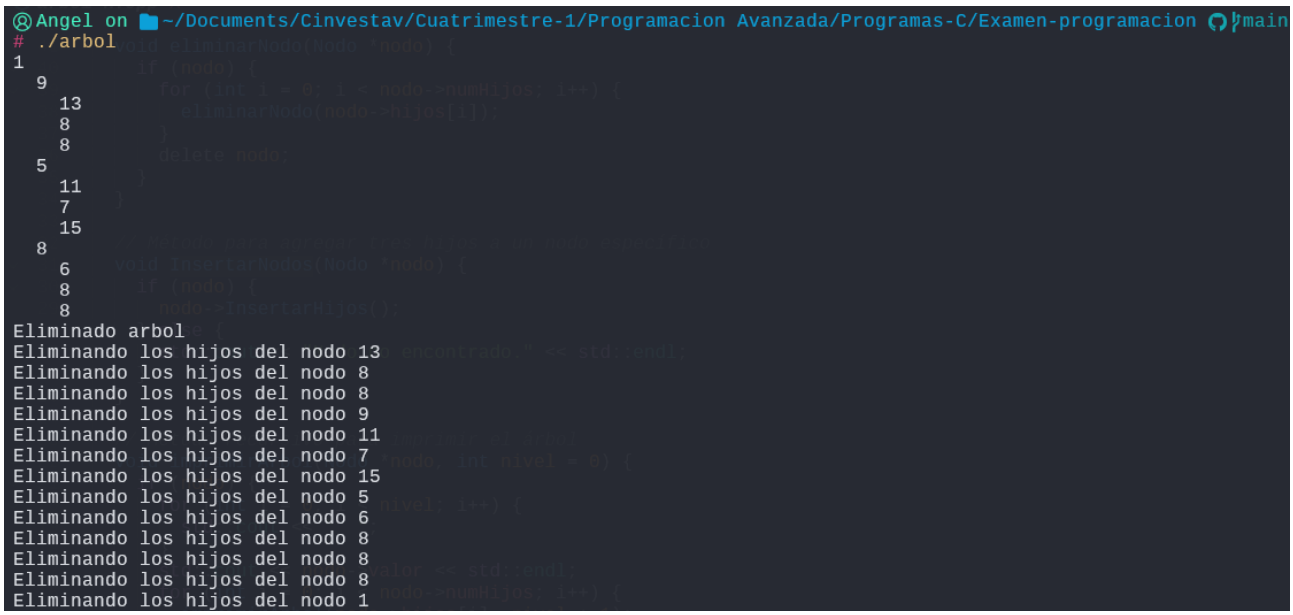


Figure 4: Primera parte de la ejecución de la lista