UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE

SEDE SANTO DOMINGO DE LOS TSÁCHILAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	202350 mayo– septiembre 2023
ASIGNATURA	:	Fundamentos de Programación
TEMA	:	Tipos de Datos Abstractos y archivos
ESTUDIANTE	:	**González Lojan Jilmar Iván**
		Maigua Toro Erick Jardiel
		Mendoza Justin
NIVEL-PARALELO - NRC:		Primero A – Nrc 12295
DOCENTE	:	Ing. Verónica Martínez C., Mgs.
FECHA DE ENTREGA	:	12 de junio del 2023

SANTO DOMINGO – ECUADOR

INDICE GENERAL

1 Introducción

Los Tipos de Datos Abstractos (TDA) desempeñan un papel fundamental en el mundo de la programación y estructuras de datos. Estos conceptos proveen una abstracción poderosa para organizar y manipular información, permitiendo a los programadores desarrollar algoritmos eficientes y facilitando la implementación de estructuras de datos complejas. Uno de los aspectos más destacados de los TDAs es su capacidad para separar la interfaz de acceso a los datos de los detalles de implementación subyacentes, lo que mejora la legibilidad del código.

2 Objetivos

2.1 Objetivo General:

El objetivo general es estudiar los Tipos de Datos Abstractos: Listas, Pilas y Colas en C++, comprendiendo sus características y aplicaciones en el desarrollo de programas eficientes.

2.2 Objetivos Específicos:

- Definir y describir las características principales de los Tipos de Datos Abstractos: Listas, Pilas y Colas.
- Analizar las operaciones básicas de cada TDA, incluyendo la declaración, acceso, almacenamiento, lectura y recuperación de información en el lenguaje de programación C++.
- Proporcionar ejemplos prácticos de implementación de cada TDA en C++ y comparar su rendimiento en diferentes situaciones.

3 Marco Teórico

3.1 Que es TDA

Un Tipo de Datos Abstracto (TDA) es una estructura de datos que define un conjunto de operaciones y su comportamiento sin especificar los detalles internos de la implementación. En otras palabras, es una forma de encapsular datos y operaciones relacionadas en una unidad coherente, ocultando la complejidad interna y proporcionando una interfaz clara y consistente para interactuar con la estructura de datos.

Las listas, pilas y colas son ejemplos comunes de TDAs utilizados para organizar y manipular datos de manera eficiente. Cada uno de ellos tiene un conjunto específico de operaciones definidas y un comportamiento bien definido.

- Listas: Una lista es una estructura de datos que almacena una secuencia ordenada de elementos. Cada elemento en la lista tiene una posición o índice único que permite acceder a él.
- Pilas: Una Pila es una estructura de datos LIFO (Last In, First Out) donde el último elemento agregado es el primero en ser retirado. Es como una pila de platos, donde solo se puede agregar o quitar elementos desde la cima.
- Colas: Una Cola es una estructura de datos FIFO (First In, First Out) donde el primer elemento agregado es el primero en ser retirado. Funciona como una cola en una tienda, donde el primer cliente que llega es el primero en ser atendido.

3.2 Declaración

La estructura que define el nodo de la pila se declara como sigue:

```
 8
 9  struct Nodo {
10      char dato;
11      Nodo* siguiente;
12  };
13
```

Aquí se define una estructura llamada `Nodo`, que contiene un miembro `dato` de tipo `char` para almacenar el valor del nodo, y un puntero `siguiente` de tipo `Nodo*` que apunta al siguiente nodo en la pila.

La pila en sí se declara en la función `MenuGonzalez()` como un puntero a `Nodo`, inicialmente apuntando a `NULL`.

```
39      char dato;
40      Nodo* aux = NULL;
41
```

3.3 Definición de variables

En la función `MenuGonzalez()` se definen las siguientes variables:

`Nodo* pila`: Es un puntero a la estructura `Nodo`, utilizado para apuntar al primer elemento de la pila (el nodo en la cima).

`char opc`: Es una variable de tipo `char` que se utiliza para almacenar la opción seleccionada por el usuario en el menú.

`char dato`: Es una variable de tipo `char` que se utiliza para almacenar el dato que se va a agregar a la pila.

`Nodo* aux`: Es un puntero a la estructura `Nodo` que se utiliza como variable auxiliar para recorrer la pila durante la operación de mostrar los elementos o eliminar la pila.

Estas variables se utilizan a lo largo de la función `MenuGonzalez()` para interactuar con la pila y proporcionar la funcionalidad del menú.

```
35
36  void MenuGonzalez() {
37      Nodo* pila = NULL;
38      char opc;
39      char dato;
40      Nodo* aux = NULL;
41
```

3.4 Acceso

```
13
14  void agregarPila(Nodo*& pila, char n) {
15      Nodo* nuevo_nodo = new Nodo();
16      nuevo_nodo->dato = n;
17      nuevo_nodo->siguiente = pila;
18      pila = nuevo_nodo;
19  }
20
21  void sacarPila(Nodo*& pila, char& n) {
22      Nodo* aux = pila;
23      n = aux->dato;
24      pila = aux->siguiente;
25      delete aux;
26  }
27
```

3.5 Almacenamiento de información

El almacenamiento de información se realiza en la función `agregarPila()`, donde se crea un nuevo nodo y se agrega al principio de la pila:

```
13
14  void agregarPila(Nodo*& pila, char n) {
15      Nodo* nuevo_nodo = new Nodo();
16      nuevo_nodo->dato = n;
17      nuevo_nodo->siguiente = pila;
18      pila = nuevo_nodo;
19  }
20
```

El nuevo nodo se crea dinámicamente en memoria usando `new`, y su miembro `siguiente` se establece para apuntar al nodo que antes estaba en la cima de la pila. Luego, se actualiza el puntero `pila` para que apunte al nuevo nodo, convirtiéndose así en el nuevo nodo superior de la pila.

3.6 Lectura de información

La lectura de información se lleva a cabo en la función `sacarPila()`, donde se extrae el valor del nodo en la cima de la pila y se almacena en la variable `n`:

```
20
21  void sacarPila(Nodo*& pila, char& n) {
22      Nodo* aux = pila;
23      n = aux->dato;
24      pila = aux->siguiente;
25      delete aux;
26  }
27
```

Primero, se crea un puntero auxiliar `aux` que apunta al nodo en la cima de la pila. Luego, el valor de este nodo se copia en la variable `n` y se actualiza el puntero `pila` para que apunte al siguiente nodo en la pila (el nodo siguiente del que se eliminó). Finalmente, el nodo en la cima se libera de la memoria utilizando `delete`.

3.7 Recuperación de información

La recuperación de información se realiza en la función `MenuGonzalez()` cuando se muestra el contenido de la pila en el caso 2:

```
59              case 2:
60                  cout << "\nMostrando todos los elementos de la pila: ";
61                  aux = pila;
62                  if (aux == NULL) {
63                      cout << "La pila esta vacia.";
64                  } else {
65                      while (aux != NULL) {
66                          cout << aux->dato;
67                          if (aux->siguiente != NULL) {
68                              cout << " , ";
69                          }
70                          aux = aux->siguiente;
71                      }
72                  }
73                  cout << "\n" << endl;
74                  system("pause");
75                  break;
```

Aquí, se utiliza un puntero auxiliar `aux` para recorrer la pila y mostrar el valor de cada nodo. El bucle `while` se repite hasta que el puntero auxiliar `aux` llega al final de la pila (cuando es `NULL`).

3.8 Entrada y salida por archivos

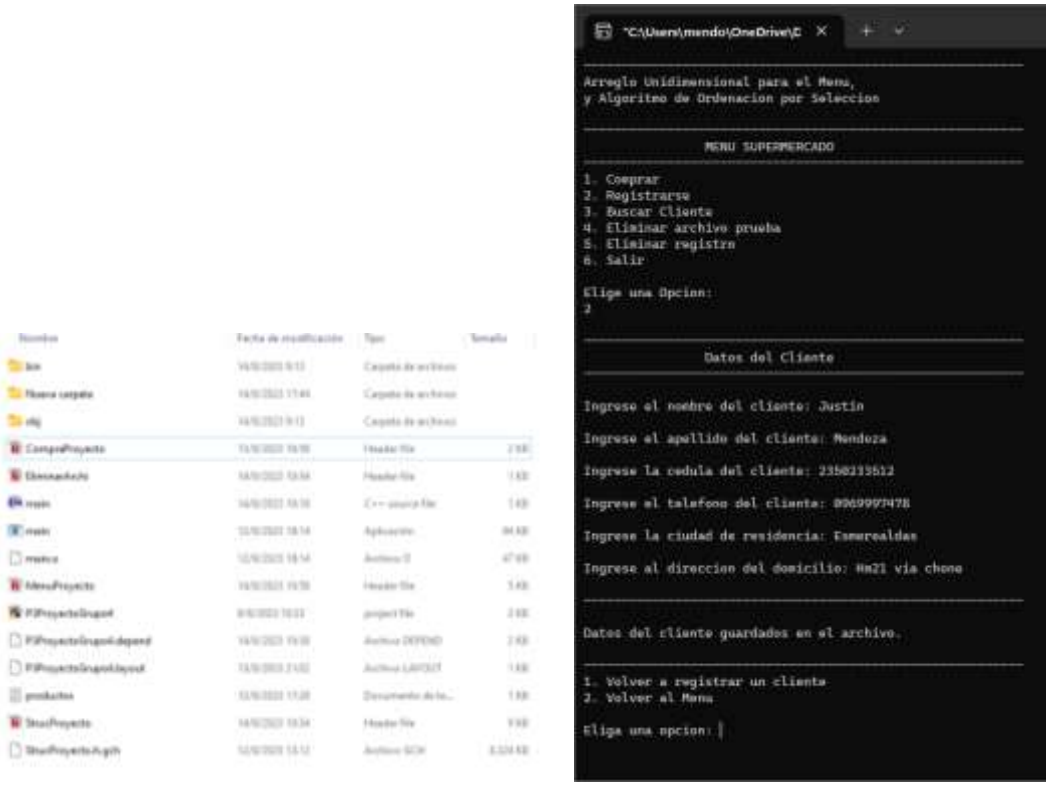
Entrada por archivos: Proceso de lectura de datos desde un archivo externo (como un archivo de texto, una hoja de cálculo, etc.) hacia un programa. Estos datos pueden ser utilizados en el programa para diversos propósitos, como cálculos, análisis, o transformaciones.

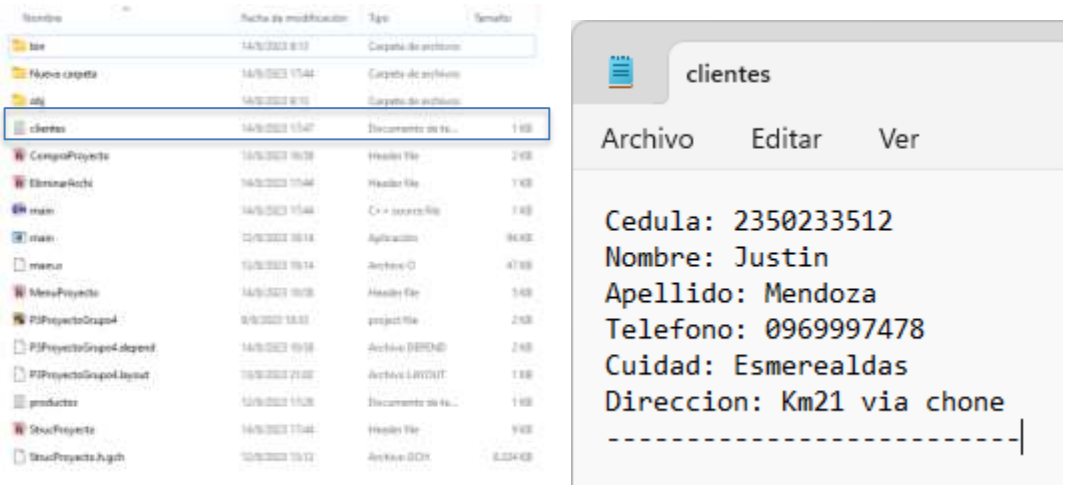
Salida por archivos: Proceso de escritura de datos desde un programa hacia un archivo externo. Tras la ejecución del programa, estos datos quedan almacenados en el archivo y pueden ser revisados, compartidos o procesados posteriormente.

Implementar operaciones CRUD (Create, Read, Update, Delete) en archivos en C++ implica utilizar la biblioteca `<fstream>`, que permite manejar archivos. A continuación, te presentaré un ejemplo simple de operaciones CRUD sobre un archivo que almacena nombres de personas, uno por línea.

1. Create: Agregar un nuevo nombre al archivo.

```
numeroCedula(cliente);   //Funcion para ingresar el numero de cedula

numeroTelefono(cliente); //Funcion para ingresar el numero de telefono

cout << "\nIngrese la ciudad de residencia: ";
getline(cin,cliente.domicilio.ciudad);

cout << "\nIngrese al direccion del domicilio: ";
getline(cin,cliente.domicilio.calle);

ofstream archivoClientes("clientes.txt", ios::app);  //Abriendo el archivo en modo agregar texto

if (archivoClientes.is_open()) {

    archivoClientes << "Cedula: " << cliente.cedula << endl;
    archivoClientes << "Nombre: " << cliente.nombre << endl;
    archivoClientes << "Apellido: " << cliente.apellido << endl;
    archivoClientes << "Telefono: " << cliente.telefono << endl;
    archivoClientes << "Cuidad: " << cliente.domicilio.ciudad << endl;
    archivoClientes << "Direccion: " << cliente.domicilio.calle << endl;
    archivoClientes << "--------------------------" << endl;

    archivoClientes.close();
    cout << "\n-----------------------------------------------------------" << endl;
```





2. Read: Leer todos los nombres del archivo.

```
void datosCliente() {

    string cedulaBuscar;
    int opc;

    do {
        cout << "\n-------------------------------------------------------------" << endl;
        cout << "\t\tBuscar Cliente" << endl;
        cout << "-------------------------------------------------------------" << endl;
        fflush(stdin);
        cout << "\nIngrese el numero de cedula: ";
        getline(cin, cedulaBuscar);

        if (esNumero(cedulaBuscar, 10)) {
            buscarCliente(cedulaBuscar);

            //cout << endl;
            //system("pause");

            do {
                cout << "\n-------------------------------------------------------------" << endl;
                cout << "1. Volver a escribir un numero de cedula " << endl;
                cout << "2. Volver al Menu" << endl;
                cout << "\nElige una opcion: ";
                cin >> opc;
                if (cin.fail() || (opc != 1 && opc != 2)) {
                    cout << "\nOpcion incorrecta" << endl;
                    cin.clear();
```

```
void buscarCliente(const string& cedula) {

    ifstream archivoClientes("clientes.txt");

    if (archivoClientes.is_open()) {

        string linea;

        bool clienteEncontrado = false;

        while (getline(archivoClientes, linea)) {
            string cadena = "Cedula: ";
            size_t pos = linea.find(cadena.append(cedula));

            if (pos != string::npos) {
                clienteEncontrado = true;
                cout << "\nCliente encontrado:" << endl;
                cout << "---------------------------" << endl;
                cout << linea << endl;  // Imprimir la línea con la cédula
                for (int i = 0; i < 5; i++) {
                    getline(archivoClientes, linea);  // Leer las siguientes 5 líneas (nombre, apellido, telefono)
                    cout << linea << endl;  // Imprimir nombre, apellido, telefono
                }
                cout << "---------------------------" << endl;
                break;
            }
        }

        if (!clienteEncontrado) {
```

```
                cout << "---------------------------" << endl;
                cout << linea << endl;  // Imprimir la línea con la cédula
                for (int i = 0; i < 5; i++) {
                    getline(archivoClientes, linea);  // Leer las siguientes 5 líneas (nombre, apellido, telefono)
                    cout << linea << endl;  // Imprimir nombre, apellido, telefono
                }
                cout << "---------------------------" << endl;
                break;
            }
        }

        if (!clienteEncontrado) {
            cout << "\nCliente no encontrado." << endl;
        }

        archivoClientes.close();
    } else {
        cout << "No se pudo abrir el archivo." << endl;
    }
}
```

```
--------------------------------------------------------------------------------
                              MENU SUPERMERCADO
--------------------------------------------------------------------------------
1. Comprar
2. Registrarse
3. Buscar Cliente
4. Eliminar archivo prueba
5. Eliminar registro
6. Salir

Elige una Opcion:
3

--------------------------------------------------------------------------------
                              Buscar Cliente
--------------------------------------------------------------------------------

Ingrese el numero de cedula: 2350233512

Cliente encontrado:
--------------------
Cedula: 2350233512
Nombre: Justin
Apellido: Mendoza
Telefono: 0969997478
--------------------

--------------------------------------------------------------------------------
1. Volver a escribir un numero de cedula
2. Volver al Menu

Eliga una opcion:
```

3. Update: Modificar un nombre existente.

```
ofstream archivoClientes("clientes.txt", ios::app);  //Abriendo el archivo en modo agregar texto
```

```
--------------------------------------------------------------------------------
                              MENU SUPERMERCADO
--------------------------------------------------------------------------------
1. Comprar
2. Registrarse
3. Buscar Cliente
4. Eliminar archivo prueba
5. Eliminar registro
6. Salir

Elige una Opcion:
3

--------------------------------------------------------------------------------
                              Datos del Cliente
--------------------------------------------------------------------------------

Ingrese el nombre del cliente: Andres

Ingrese el apellido del cliente: dueñas

Ingrese la cedula del cliente: 1234567890

Ingrese el telefono del cliente: 0123456789

Ingrese la ciudad de residencia: El carmen

Ingrese el direccion del domicilio: venado

--------------------------------------------------------------------------------

Datos del cliente guardados en el archivo.

--------------------------------------------------------------------------------
1. Volver a registrar un cliente
2. Volver al Menu

Eliga una opcion:
```

```
Cedula: 2350233512
Nombre: Justin
Apellido: Mendoza
Telefono: 0969997478
Cuidad: Esmerealdas
Direccion: Km21 via chone
--------------------------
Cedula: 1234567890
Nombre: Andres
Apellido: dueñas
Telefono: 0123456789
Cuidad: El carmen
Direccion: venado
--------------------------
```

4. Delete: Eliminar un nombre del archivo.

```
void EliminarArchivo(){
if(remove("prueba.txt")==0){
    cout<< "El archivo se borro con exito..." << endl;
}else{
    cout<< "No se encontro el archivo..." << endl;
}
}
#endif // ELIMINARARCHI_H_INCLUDED
```

3.9 Archivos de texto

Un archivo de texto es un tipo de archivo informático que contiene texto estructurado de forma secuencial y legible por humanos, sin información adicional de formato más allá de caracteres básicos como espacios, tabuladores y saltos de línea. A menudo se utilizan para almacenar datos como código fuente, configuraciones, documentos y notas, y pueden ser leídos y editados con editores de texto simples.

- open() : Se usa para abrir un archivo.

```
void open(const char* filename, ios_base::openmode mode);
```

```
ofstream outfile;
outfile.open("example.txt", ios::out);
```

- close() : Se usa para cerrar un archivo después de su uso.

```
void close();
```

```
outfile.close();
```

3.10 Archivos Binarios

Un archivo binario es un archivo que contiene datos en un formato no textual, es decir, datos representados en formato binario que no se limitan a caracteres legibles. A diferencia de los archivos de texto, que contienen texto en caracteres que generalmente son humanamente legibles, los archivos binarios pueden contener cualquier tipo de dato, como imágenes, audio, video, programas ejecutables o estructuras de datos específicas. Su lectura y escritura requiere programas o herramientas específicas capaces de interpretar correctamente su contenido.

- write() : Escribe una secuencia de bytes en un archivo binario.

```
ostream& write(const char* s, streamsize n);
```

- read() : Lee una secuencia de bytes de un archivo binario.

```
istream& read(char* s, streamsize n);
```

4 Conclusiones

Los Tipos de Datos Abstractos: Listas, Pilas y Colas, son herramientas esenciales en la programación en C++. Cada uno ofrece un enfoque único para organizar y manipular datos, lo que los hace ideales para diferentes escenarios. Las operaciones de declaración, acceso, almacenamiento, lectura y recuperación proporcionan una manera eficiente de trabajar con estos TDAs en C++. La elección adecuada del TDA depende de la aplicación específica y los requerimientos del problema a resolver.

5 Recomendaciones

Considera las necesidades del problema y las operaciones requeridas al elegir entre Listas, Pilas o Colas.

Asegúrate de usar los métodos adecuados para el acceso y modificación de los datos en cada TDA.

Prueba tus implementaciones con casos de prueba para verificar su correcto funcionamiento.

6 Bibliografía

Encarnación, W. M. S. (2013, junio 3). *Pilas, colas, y listas estructura de datos*. Slideshare.net. https://www.slideshare.net/diwal10/pilas-colas-y-listas-estructura-de-datos'

95. Programación en C++ || Pilas || Concepto de Pila. (2016, septiembre 14). YouTube. https://www.youtube.com/watch?v=joAw2jWgZqA&list=PLWtYZ2ejMVJlUu1rEHLC0i_oibctkl0Vh&index=96

99. Programación en C++ || Colas || Concepto de Cola. (2016, octubre 14). YouTube. https://www.youtube.com/watch?v=5CClpYQTGUI&list=PLWtYZ2ejMVJlUu1rEHLC0i_oibctkl0Vh&index=101

103. Programación en C++ || Listas || Concepto de Lista Enlazada. (2016, octubre 23). YouTube. https://www.youtube.com/watch?v=15urP2LmfqY&list=PLWtYZ2ejMVJlUu1rEHLC0i_oibctkl0Vh&index=104

Muñoz, J. I. G. (s/f). *Implementación de Pilas y Colas en C*. Scribd. Recuperado el 6 de agosto de 2023, de https://es.scribd.com/doc/39509324/Implementacion-de-Pilas-y-Colas-en-C

7 Legalización de documento

Nombres y Apellidos: González Lojan Jilmar Ivan

CI: 1752597953

Firma:

Nombres y Apellidos**:** Erick Jardiel Maigua Erick

CI: 0150327260

Firma:

Nombres y Apellidos: Mendoza Justin

CI: 2350233512

Firma: