Logotipo

Descripción generada automáticamente con confianza media

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**FACULTAD DE INGENIERÍA**

**CARRERA DE INGENIERÍA DE SOFTWARE**

**CARRERA DE CIENCIAS DE LA COMPUTACIÓN**

**CARRERA DE INGENIERÍA DE SISTEMAS DE LA INFORMACIÓN**

**CICLO 2022-01**

**ALGORITMOS Y ESTRUCTURAS DE DATOS – CC182**

**Sección SV31**

**INFORME DE TRABAJO FINAL**

**“GESTIÓN DE TAREAS”**

**PROFESOR DEL CURSO**

Edson Duilio Mendiolaza Cornejo

**INTEGRANTES**

|  |  |  |
| --- | --- | --- |
| **NOMBRE** | **CÓDIGO** | **CARRERA** |
| Arroyo Ormeño, André Alonso | U202114714 | Ingeniería de Software |
| Comettant Rubiños, Jessica Elizabeth | U20211C009 | Ciencias de la Computación |
| Huarcaya Chavez, Miguel Alejandro Daniel | U202116207 | Ingeniería de Software |

**Enlace de github:**

<https://github.com/rdulanto/AED20221-G07.git>

**FECHA DE ENTREGA MÁXIMA**

19/06/2022

**SEDE MONTERRICO-VILLA, junio del 2022**

**Índice**

1. [Introducción 3](#_Toc99961179)
2. [Objetivos 4](#_Toc99961179)
3. Explicación del caso de estudio………...….....………………….………………5
4. Explicación de estructuras.................………………...………………………….5
5. Estudio de la complejidad de notación ………………………………………….5
6. [Diagrama de clases………………………………………………………………8](#_Toc99961179)
7. [Fundamento teórico del código 9](#_Toc99961179)
8. [Participación del equipo 11](#_Toc99961179)
9. [Anexos 12](#_Toc99961179)
10. **INTRODUCCIÓN**

El presente trabajo busca evidenciar el aprendizaje adquirido por las/los estudiantes del curso de Algoritmos y Estructuras de datos. Ello se demostrará a través de la implementación de un gestor de tareas en el programa Visual Studio 2019 mediante el uso del lenguaje de programación c++. Las técnicas de programación utilizadas enfatizan el uso de los *tipos de datos abstractos, la complejidad asintótica de algoritmos y algoritmo recursivo*. Asimismo, se maneja diferentes tipos de estructuras de datos como *vectores, colas, listas y pilas*. Todo esto con la finalidad de alcanzar el logro del curso y las competencias postuladas por la universidad (Pensamiento Innovador y Diseño y desarrollo de una solución) con relación a la carrera de Ciencias de la Computación e Ingeniería de Software.

1. **OBJETIVOS**

Para Ciencias de la Computación:

* Diseñar, implementar y evaluar una solución basada en la computación para cumplir con un conjunto de requisitos computacionales en el contexto de la disciplina del programa.

Para Ingeniería de Software:

* Capacidad de aplicar el diseño de ingeniería para producir soluciones que satisfagan necesidades específicas con consideración de salud pública, seguridad y bienestar, así como factores globales, culturales, sociales, ambientales y económicos.

1. **EXPLICACIÓN DEL CASO DE ESTUDIO**

A través del presente trabajo, se ha realizado la emulación básica de una aplicación encargada de gestionar diferentes tópicos de tareas. El nombre de este aplicativo es Microsoft To Do, conocida a nivel mundial por la organización y planificación que ofrece a sus usuarios. En este escenario, la funcionalidad del programa busca cumplir con los mismos requisitos funcionales y algoritmos de la aplicación. El equipo de trabajo implementó diversos procesos similares como es el almacenamiento de tareas, eventos, contactos, entre otros. De igual forma, adicionó otras funcionalidades como es el registro de canciones y un método de estudio atractivo para el usuario que parte de la inserción de preguntas y respuestas en el programa.

1. **EXPLICACIÓN DE ESTRUCTURAS**

El equipo de trabajo manejo el uso de diferentes tipos abstractos de datos (TDA) lineales para agilizar el almacenamiento de la información del usuario. Entre ellos tenemos a las pilas, colas, listas y vectores. Los vectores nos facilitan el acceso a los objetos de información a través del método *at()* cuya complejidad asintótica es de O(n). Ello nos permitió mostrar en la consola los datos registrados a partir de archivos binarios. Por otra parte, las listas simplificaron el proceso de ordenamiento avanzado de los contactos registrados mediante el método *radixsort*(). Adicional a ello, para la implementación de la *clase Quizz(),* se empleó el desordenamiento. Finalmente, el uso de pilas y colas se ejecutó para otorgar mayor dinamismo al código.

1. **ESTUDIO DE LA COMPLEJIDAD DE NOTACIÓN**

* **PILA *(class Stack):***
* **Método Push\_Back():**

void push\_back(Generic n) {

Node4<Generic>\* nodito = new Node4<Generic>(n);

nodito->next = first;

first = nodito;

last = first;

siz++;

}

* ***Big(O)= O(1) Tiempo constante***
* **Método At ():**

Node4<Generic>\* at(int i) {

Node4<Generic>\* aux = first;

for (int j = 0; j < i; j++) {

aux = aux->next;

}

return aux;

}

* ***Big(O)= O(n) Tiempo lineal***
* **Método print ():**

void print(){

Node4<Generic>\* aux = first;

for (int i = 0; i < siz; i++) {

mostrar(aux->elem);

aux = aux->next;

}

}

* ***Big(O)= O(n) Tiempo lineal***
* **Cola *(class Queue):***
* **Método print ():**

void print() {

if (siz > 0) {

Node<Generic>\* aux = head;

while (aux->next != nullptr) {

mostrar(aux->date);

aux = aux->next;

}

if (aux->next == nullptr) {

mostrar(aux->date);

}

cout << endl;

}

}

* ***Big(O)= O(1) Tiempo constante***
* **Método enqueue ():**

Node<Generic>\* nuevo = new Node<Generic>(v);

nuevo->next = nullptr;

if (empty()) {

head = nuevo;

tail = nuevo;

}

else {

tail->next = nuevo;

}

tail = nuevo;

cout << endl;

siz++;

}

* ***Big(O)= O(1) Tiempo constante***
* **Lista *(class List):***
* **Método print():**

void \_print(Node\* n) {

if (n == nullptr) return;

disorder();

mostrar(n->value);

\_print(n->next);

}

* ***Big(O)= O(1) Tiempo constante***
* **Método disorder():**

void disorder() {

for (int i = size; i > 0; i--) {

Node\* aux = first; Node\* aux1 = first;

int j = rand() % i;

for (int k = 0; k < j; k++)

aux = aux->next;

for (int l = 0; l < i - 1; l++)

aux1 = aux1->next;

swap(&aux1->value, &aux->value);

}

}

* ***Big(O)= O(n2) Tiempo cuadrático***
* **Método pushBack():**

void pushBack(Generic value) {

if (size == 0) {

Node\* n = new Node(value);

first = n;

last = n;

++size;

}

else {

Node\* n = new Node(value);

last->next = n;

last = n;

++size;

}

}

* ***Big(O)= O(1) Tiempo constante***
* **Método mergesort():**

void \_mergeSort(int i, int f) {

if (i == f)

return;

size\_t middle = (i + f) / 2;

\_mergeSort(i, middle);

\_mergeSort(middle + 1, f);

merge(i, middle, f);}

* ***Big(O)= O(1) Tiempo constante***
* **Método radixsort():**

void radixsort() {

long max = getMax();

for (long place = 1; max / place > 0; place \*= 10)

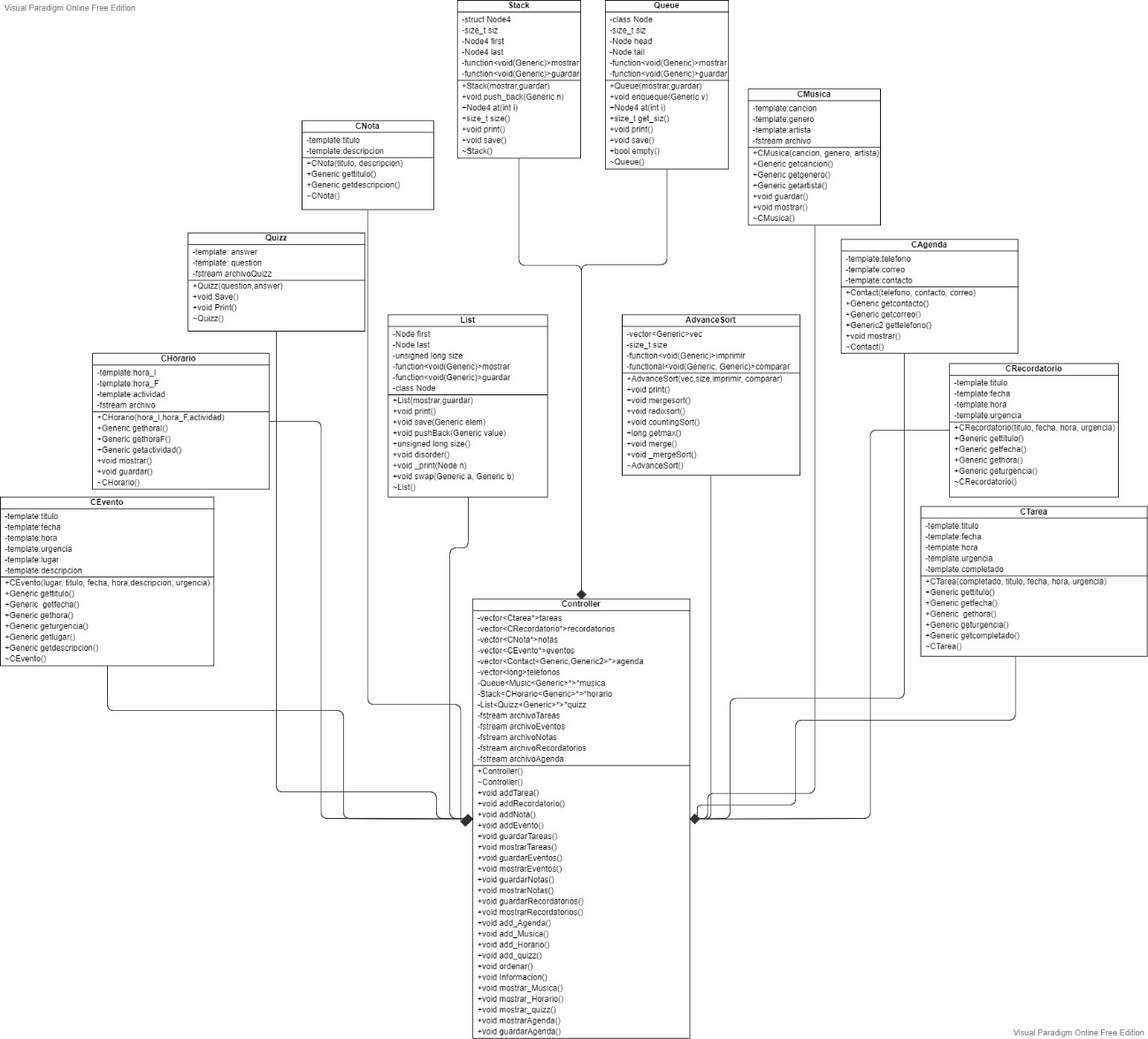
countingSort(place);

}

* ***Big(O)= O(n) Tiempo lineal***

1. **DIAGRAMA DE CLASES**

*Fig. 01 Diagrama de clases de la implementación del código*

****

Fuente: Elaboración Propia con la herramienta Visual Paradigm.

*Nota: En el diagrama, se visualiza la cantidad de entidades, métodos y tipo de variables empleadas por el equipo de trabajo en el desarrollo de programa. Se resalta el uso de la programación genérica a través de las plantillas (templates).*

1. **FUNDAMENTO TEÓRICO DEL CÓDIGO**

* **USO DEL PARADIGMA POO (PROGRAMACIÓN ORIENTADA A OBJETOS)**

El equipo de trabajo empleó los principios fundamentales de Abstracción y Encapsulamiento para la creación de atributos y métodos en cada entidad. A través del *encapsulamiento*, las clases *CNota, CTarea, CRecordatorio, CEvento, CAgenda, CMusica, CHorario, CQuizz y Controller* fueron capaces de agrupar los métodos (*gets*), variables requeridas en la implementación (*título, descripcion, fecha, hora, urgencia, lugar, completado, correo, contacto, telefono, cancion, genero, artista, actividad, horaI y horaF* ) y conjuntos dinámicos como *Pilas(estructuras LIFO), Colas(estructuras FIFO) y Listas Sencillas,* lo que nos permitió incrementar la cohesión del sistema*. Public* nos permitió acceder a los datos de forma abierta, mientrasque *private* otorgaba seguridad a los datos propios de cada clase. Asimismo, la *abstracción* nos facilitó la representación de los atributos y métodos necesarios para lograr concretar la creación de la *app* minimalista de *Microsoft To Do.* Ello convierte a la clase en un nivel superior de abstracción.

* **USO DE TEMPLATES:**

El equipo de trabajo aplicó el uso de *templates* (título, descripcion, fecha, hora, urgencia, lugar, completado, correo, contacto, telefono, cancion, genero, artista, actividad, horaI y horaF) como tipo de dato abstracto fundamental para toda la implementación del código. Gracias a este *TDA,* se evitó la sobrecarga de funciones y variables para diferentes tipos de datos como *int,long, string,* entre otros. Ello simplificó el espacio que emplea el código (programación genérica).

* **USO DE LAMBDAS**

El equipo de trabajo implementó la función anónima *lambda* en la estructura principal del código (*main*) con la finalidad de utilizar las funciones *mostrarTareas(),* *mostrarEventos(), mostrasNotas()*, *mostrarRecordatorios(), mostrar\_Musica(), mostrar\_Agenda() y mostrar\_Horario().* Esto permitióvisibilizar en la consola la información insertada por el usuario en dependencia del tipo de apunte seleccionado. Asimismo, se ejecutó el uso de punteros a función y lambdas para los métodos print(), *mergesort(), radixsort() y save().*

* **USO DE ESTRUCTURAS DE DATOS**

Las estructuras de datos empleadas por el equipo de trabajo fueron *vectores, pilas, colas y listas sencillas*. A través de la librería *vector*, se accedió a métodos (.pushback(), .erase()) que posibilitaron un manejo más dinámico de la memoria. Con respecto al uso de listas, colas y pilas; cada estructura lineal se mostraba compuesta de *Nodos*, elementos que almacenaban la información ingresada por el usuario. Estas estructuras se visualizan en las entidades *CAgenda(), CMusica(), CTarea(), CNotas(), CRecordatorios(), CEventos(), Quizz() y CHorario().*

* **USO DE ORDENAMIENTOS**

El ordenamiento implementado es perteneciente al algoritmo *“Fisher-Yates shuffle”,* el cual consiste en recorrer una estructura y obtener un desordenamiento. Se implementó *Fisher* en la clase *List()*, entidad creada con una estructura *LIFO*, con el objetivo de obtener, de forma aleatoria, preguntas y respuestas ingresadas por el usuario. Del mismo modo, se manejo el uso de ordenamientos avanzados *radixsort() y mergesort()* en el registro de números telefónicos de la clase *CAgenda().*

1. **PARTICIPACIÓN DEL EQUIPO**

|  |  |  |
| --- | --- | --- |
| ACTIVIDADES | RESPONSABLE | AUTOVALORACIÓN |
| * Realización del informe. * Colaboración con el código del main.cpp y la clase *Controller*. * Creación de las entidades *CNota, CTarea, CRecordatorio, CEvento, CAgenda, CMusica, CHorario, CQuizz ,Queue ,List, Stack y AdvanceSort* | Arroyo Ormeño, André Alonso | *33.3%* |
| * Realización del informe * Colaboración con el código del main.cpp y la clase *Controller* * Creación de las entidades *CNota, CTarea, CRecordatorio, CEvento, CAgenda, CMusica, CHorario, CQuizz, ,Queue ,List, Stack y AdvanceSort* | Comettant Rubiños, Jessica Elizabeth | *33.3%* |
| * Realización del informe. * Colaboración con el código del main.cpp y la clase *Controller*. * Creación de las entidades *CNota, CTarea, CRecordatorio, CEvento, CAgenda, CMusica, CHorario, CQuizz, ,Queue ,List, Stack y AdvanceSort* | Huarcaya Chavez, Miguel Alejandro Daniel | *33.3%* |

1. **ANEXOS**

*Fig. 02 Diagrama de flujo inicial del código*

Diagrama

Descripción generada automáticamente

*Nota: El siguiente diagrama representa el prototipo inicial de la implementación del código.*