INFORME DE PROGRAMACION: PARCIAL N°2

GRUPO 1:

- Vargas, Rodrigo Sebastián
- Saillen, Simón

MODALIDAD: Virtual

¿Cómo es la idea principal?

Empezamos con la idea de crear una clase *class Post*, esta sería la estructura base, de la cual luego el árbol basara sus nodos, en esta clase se guarían todos los parámetros importantes para los métodos del árbol, teniendo el tipo de *objeto Post* (2 si es un post, 1 si es un comentario y 0 si es una respuesta), los id correspondientes, el título, contenido, etc.

Después de algunos ajustes y correcciones llegamos a la clase que ven abajo:

```
class Post {
private:
  int tipo; // Post > Comentario > Respuesta == 2 > 1 > 0
  string titulo;
  string nombre;
  Tiempo t;
  string contenido;
  int rating;
  int idPost;
  int idComentario;
  int idRespuesta;
  int participaciones;
```

Imagen 1: “Parámetros privados de la clase Post”

En los parámetros public pusimos todos los *setters* y *getters* necesarios, (no los mostramos porque son bastantes y ocupan mucho espacio en la hoja del informe).

Aunque una vez terminamos con la idea de la clase Post nos encontramos con otra complejidad, el Árbol Binario que vimos en clase tipo *Template* no nos servía, debido a que no podíamos usar los métodos *setter* ni *getter* de la clase Post, por lo que optamos por dejar la estructura base igual, pero modificar los métodos de la clase árbol así como también agregarle parámetros que vimos necesarios al momento, por ende, sigue siendo un árbol binario, solo que ahora es un árbol binario tipo Post (que por consigna funciona simulando un árbol n-ario).

Tuvimos en cuenta lo aprendido en las clases, y dejamos los métodos que hagan uso de la raíz en la parte privada de la clase Árbol. Por ende, también hay un método de nombre similar al su parte privada que solo se encarga de llamarla con la raíz en parámetro.

```
class Arbol {
private:
  Nodo<Post> *raiz, *q;
  Cola<Post> colaaux;
  Cola<Post> colaFinal;
  int Promedio = 0;
  int MAXRating = 0;
  int idMaxRat = 0;
  int Veces = 0;
  string Participe;
  void ArbolBusq(Post x, Nodo<Post> *&nuevo);
  void show(Nodo<Post> *aux, int n);
  void borrar(Nodo<Post> *&p, int id);
  void listar(Nodo<Post> *&aux, int id, int tipo);
  void VotacionAux(Nodo<Post> *&aux, int post, int com, int voto);
  void Stats(Nodo<Post> *&aux, int post);
  bool EsVacio(Nodo<Post> *&aux) { return aux == NULL; }
  int part(Nodo<Post> *&aux, string nombre, int post);
  void colasTotales(Nodo<Post> *&aux);
```

```
public:
  Arbol() { raiz = NULL; };
  //~Arbol();
  void CreaArbolBus(Post x) { ArbolBusq(x, raiz); }
  void VerArbol() { show(raiz, 0); }
  void Borrar(int idp) { borrar(raiz, idp); }
  void ListarA(int idp, int tipo) { listar(raiz,idp,tipo); }
  void MostrarLista();
  void Votacion(int post, int com, int vot) { VotacionAux(raiz, post, com, vot); }
  void Estadisticas(int post) {
    this->Veces = 0;
    Stats(raiz, post);
  }
  bool arbol_vacio() { return EsVacio(raiz); }
  int MAXParticipe(string nombre, int i) { return part(raiz, nombre, i); }
  Cola<Post> cola_Final() {
    colasTotales(raiz);
    return colaFinal;
  }
};
```

Imágenes 2 y 3: “Clase Árbol <Tipo Post>”

también como se observa, agregamos 2 colas a la clase árbol que nos servirán más adelante (una de ellas se usa para mostrar los comentarios de un usuario en distintos posts en orden cronológico, y la otra para mostrar un post en específico junto a sus comentarios).
Como la clase árbol posee bastantes métodos privados, solo vamos a mostrar la lista con su descripción breve de que hace cada una:

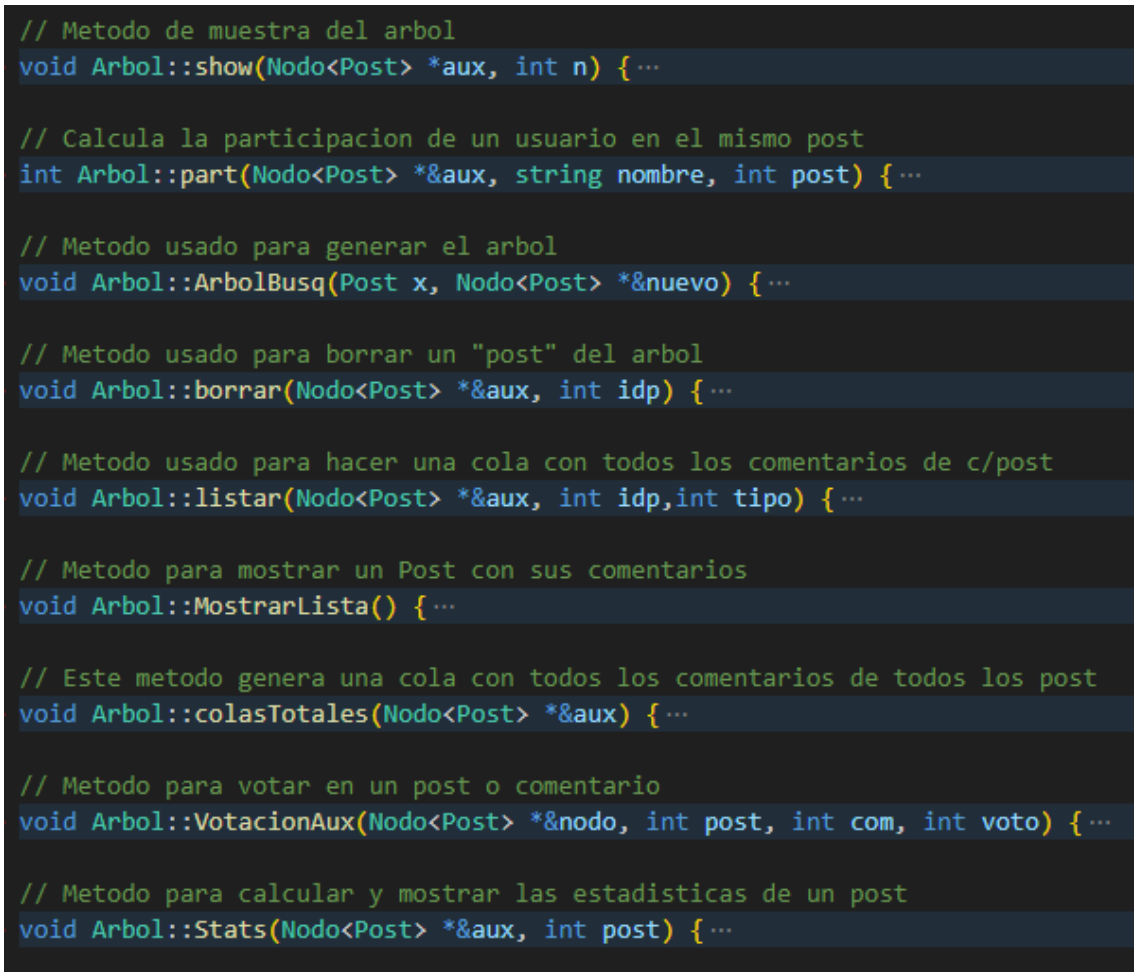

Imagen 4: “Métodos privados clase Árbol”

Los métodos más importantes son *ArbolBusq()* y *borrar*(), el primero tiene una estructura similar al método visto en clase, se llama recursivamente a si mismo hasta que se encuentra en un nodo NULL y ahí crea el nuevo “Post”, para que haga eso de manera correcta creamos el parámetro Tipo, entonces le pedimos al método que, depende del tipo que se ingrese en parámetro recorra el árbol de una forma específica, por consigna, todos los posts van uno a la derecha del otro (con el primer post siendo la raíz), y cada primer comentario a cada post va hacia la izquierda del nodo del post al que comentan (si hay mas de 1 comentario por post estos van uno a la derecha del otro), y lo mismo ocurre con las respuestas, la primera respuesta a un comentario va a la izquierda del mismo y si hay mas de una va hacia la derecha del mismo, quedando una estructura similar a la dibujada (ver Imagen 5). De manera opuesta funciona el método borrar(), primero encuentra el post que se desea borrar, y luego recorre cada comentario, y en cada comentario recorre cada respuesta hasta llegar a la última, entonces hace un borrado lógico y físico de la misma y vuelve

hacia donde estaba antes para así ir borrando cada respuesta de cada comentario, luego cada comentario y finalmente el post.

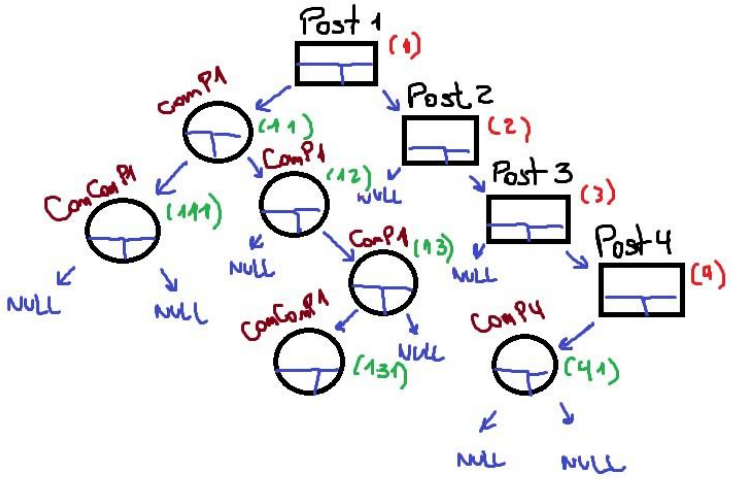

Imagen 5: “Bosquejo de la estructura del Árbol”

¿Entonces como quedaron las clases?

Tuvimos que crear una clase class GestionPost para que sea la encargada de inicializar el árbol, y tener en si misma todos los métodos necesarios para controlar el árbol, así mismo tuvimos que crear la clase Tiempo, esta es una clase que utiliza la librería *<ctime>,* el único uso que le damos a esta clase es para controlar el tiempo de los Post y Comentarios, ya que por consigna debemos tener un método para ordenar por tiempo de publicación.

```
class GestionPost {
private:
  Arbol arbol_post;
  int idPost;
  int idComentario;
  int idRespuesta;
  Post arreglo[MAXIMO];
  void ordenaQS(Post v[], int primero, int ultimo);
```

Imagen 6: “Clase GestionPost (parámetros privados)”

```
class Tiempo { // Puede ocurrir que al agregar forzosamente en main()
               // post/com/res no se guarde correctamente la hora.
private:
  time_t currentTime;
  tm *localTime;
  int anio;
  int mes;
  int dia;
  int hora;
  int min;
  int sec;
  double suma;
```

Imagen 7: “Clase Tiempo (parámetros privados)”

En el caso de la clase Tiempo no mostramos los métodos públicos porque no son de mucha importancia, lo único que rescatamos es que asigna a cada post la hora actual (al momento de la creación). En el caso de GestionPost mostraremos un pantallazo de los métodos públicos, el que más destacamos es Menu(), un método que creamos para que el usuario tenga un control mas prolijo de los otros métodos.

```
// Metodo para mostrar la participacion en distintos post de un usuario
void GestionPost::Participaciones(string user) {···

// Este metodo se encarga de llamar al metodo Quick Sort con la Cola de todos
// los comentarios
void GestionPost::ColaFinal() {···

// Version sin cout del metodo ColaFinal()
void GestionPost::Ordenador() {···

// Metodo Quick Sort visto en clase
void GestionPost::ordenaQS(Post v[], int primero, int ultimo) {···

// Metodo de interfaz de usuario para Votar
void GestionPost::Votar() {···

// Metodo de interfaz de usuario para Postear/Comentar/Responder
void GestionPost::Menu() {···

// Metodo individual para crear Post
void GestionPost::agregarPost(string titulo, string nombre, string contenido) {···

// Metodo individual para comentar posts
void GestionPost::agregarComentario(int idp, string nombre, string contenido) {···

// Metodo individual para responder comentarios
void GestionPost::agregarRespuesta(int idc, int idp, string nombre, string contenido) {···
```

Imagen 7: “Métodos públicos de la clase GestionPost”

Como se observa en la foto anterior, optamos por usar el algoritmo de ordenamiento Quick Sort (brindado en clase), para usarlo, primero usamos un método llamado *ColaFinal(),* este se encarga de llamar a un método privado del Árbol, donde recorrerá por cada post agregando a una cola (encolando) cada comentario, para luego traducir cada nodo de la cola en un Arreglo que definimos. Esto es así porque no pudimos usar el algoritmo con la cola sola (se perdían los nodos y no funcionaba el ordenamiento), por lo que optamos por esta solución, dejando *casi* intacto el método de ordenamiento visto en clase.

```
// Metodo Quick Sort visto en clase
void GestionPost::ordenaQS(Post v[], int primero, int ultimo) {
  int i, j, k;
  Post pivot, aux;

  if (ultimo > primero) {
    pivot = v[ultimo];
    // printf("\n -> %d  %d <-%4i",primero,ultimo,pivot);
    i = primero - 1;
    j = ultimo;
    for (;;) {
      while (v[++i].getTiempo().getSuma() < pivot.getTiempo().getSuma())
        ;
      while (v[--j].getTiempo().getSuma() > pivot.getTiempo().getSuma())
        ;
      if (i >= j)
        break;
      aux = v[i];
      v[i] = v[j];
      v[j] = aux;
    } // fin for
    aux = v[i];
    v[i] = v[ultimo];
    v[ultimo] = aux;
    // for(k=0;k<10;k++)printf("\n a[%d]=%d",k,v[k]);
    // printf("\n ------------------------------------");
    ordenaQS(v, primero, i - 1);
    ordenaQS(v, i + 1, ultimo);
    // printf("\nRETORNO -> %d  %d <-",primero,ultimo);
  } // fin if
}
```

Imagen 8: “Algoritmo de ordenamiento Quick Sort”

¿Qué otra dificultad encontraron?

La mayor dificultad fue el código en general, como van a observar al ver el código, nos vimos en la necesidad de utilizar muchas condiciones *if else* anidadas, generando una complejidad extra en el entendimiento y corrección del código, pero nos ayudó bastante para entender bien la recursividad necesaria en los árboles, y muchos métodos los logramos hacer bastante más rápido de lo que esperábamos.

```
int main() {

  GestionPost *gestionPost = new GestionPost();
  gestionPost->Menu();

  return 0;
}
```

```
          Geronimo[P 3]
     Franca[P 2]
Tomas[P 1]
          Julian[C 2]
               Uriel[r]
     Federico[C 1]

----- Red Social REDDIT-LIKE -----
1. Agregar un nuevo post
2. Agregar un nuevo comentario
3. Agregar una respuesta a un comentario
4. Votar en un Post/Comentario
5. Mostrar estadisticas de un Post
6. Borrar un post
7. Buscar comentarios de un Usuario
8. Mostrar todos los comentarios en orden
9. Abrir Post
0. Salir

Ingrese su opción:
```

Imagen 9 y 10: “main() y ejemplo de ejecución del programa”

Como se ve en la imagen, al delegar el control del usuario a la función Menu() logramos reducir la cantidad de código en la función *main()*, en la imagen previamente se cargaron los Post, junto a unos comentarios y una respuesta en modo de ilustración, pero aclaramos que todo es perfectamente funcional y todas las 9 elecciones del menú cumplen su funcion.