



UNIVERSIDAD CATÓLICA DE CÓRDOBA

Facultad de Ingeniería

Carrera de Licenciatura en Bioinformática

REGISTRO DE PROCEDIMIENTO DE LA REALIZACIÓN DE APLICACIÓN INFORMÁTICA PARA LECTURA “CSV”

Segundo parcial Programación III

Por

Folco, Juan Ignacio - 1912673

Tzvir, Vera Estefanía - 1913879

Córdoba, Argentina

ÍNDICE

<i>INTRODUCCIÓN.....</i>	<i>3</i>
<i>DESARROLLO.....</i>	<i>3</i>
06/11: COMIENZO DEL PROYECTO.....	3
16/11: CONSTRUCCIÓN DE LA CLASE Y REALIZACIÓN DE LOS DEMÁS ARGUMENTOS.....	6
17/11: SOLUCIÓN DE PROBLEMAS Y ARMADO DE ARGUMENTO "CASOS_EDAD.H"	12
18/11: SOLUCIÓN DE PROBLEMAS Y DESARROLLO DE "CASOS_EDAD.H" Y "P_CASOS.H"	16
19/11:AÑADIR DETALLES A MERGESORTNUM	20
20/11: LECTURA DE RANGO ETARIO	21
21/11: FINALIZACIÓN DE ARGUMENTOS	21
<i>CONCLUSIÓN.....</i>	<i>26</i>

INTRODUCCIÓN

En este segundo parcial de la materia, se llevará a cabo la utilización de la herramienta CLion para realizar un visualizador de datos de los casos de COVID-19 presentados en Argentina, siendo estos parte de un archivo .CSV, que forma parte de un dataset del ministerio de salud que se actualiza día a día.

Para ello, se lleva a cabo la realización de una aplicación informática capaz de la lectura del archivo y que, a través de línea de comandos, se acceda a la lectura de la información y el acceso a contenidos específicos, tales como información estadística, infectados y fallecidos en cada provincia, el análisis de pacientes en ciertas fechas de cuidados intensivos, entre otros tópicos.

Con esto, se busca aplicar contenido visualizado en la materia, además de formar una herramienta que sea útil al momento del acceso y lectura de datos, ya que en un principio por longitud, tamaño y falta de un orden específico puede ser complicado para hacerlo.

DESARROLLO

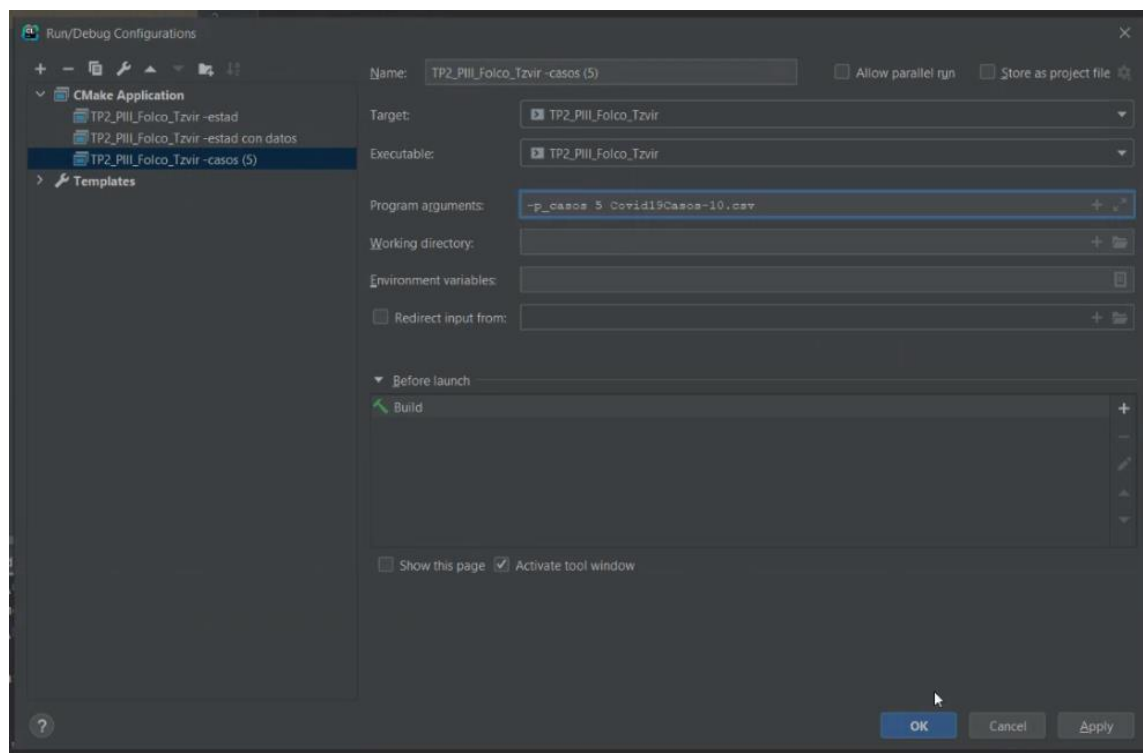
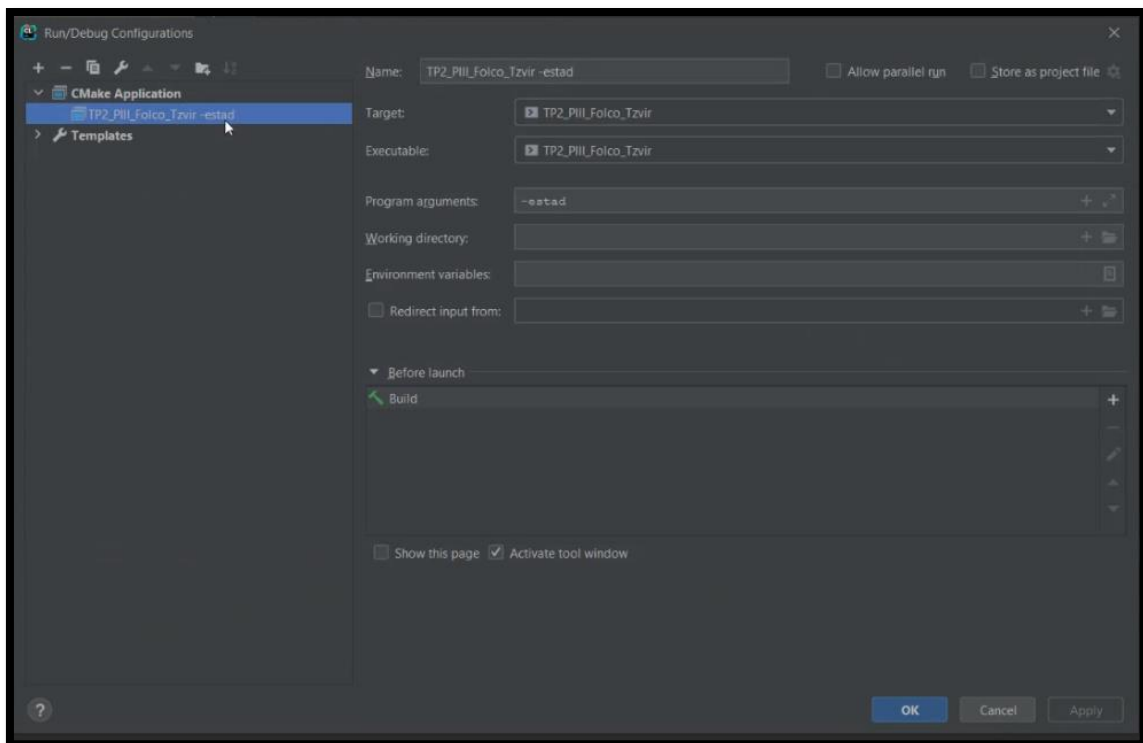
06/11: COMIENZO DEL PROYECTO

Previamente realizado el repositorio, se procede a la copia de los archivos .h de las estructuras de datos vistas en clase.

Acto seguido, se lleva a cabo la realización del main que contendrá los argumentos del proyecto (anexándolos al proyecto para su compilación y debug), siendo los siguientes:

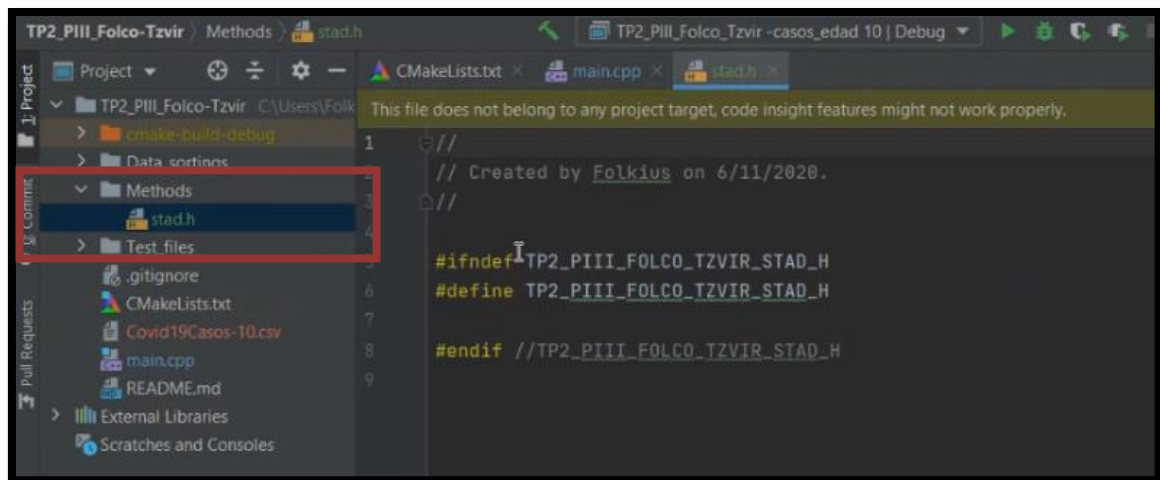
- -p.casos: en caso del pasaje de un valor, se muestra las provincias con mayor cantidad de casos ordenadas de más a menos. Si no presenta parámetros, se detalla la información relacionada a todas las provincias
- -stad: exhibición de la información estadística de cada caso.
 - ✓ Cantidad total de muestras.
 - ✓ Cantidad total de infectados.
 - ✓ Cantidad de fallecidos.
 - ✓ % de infectado por muestras.
 - ✓ % de fallecidos por infectados.
 - ✓ Cantidad de infectados por rango etario (rango de 10 años)
 - ✓ Cantidad de muertes por rango etario (rango de 10 años)
- -p_muertes: si hay parámetro, se visualiza las provincias con mayor cantidad de muertes de más a menos. En caso contrario, habrá datos de todas las provincias.
- -casos_edad: siendo ordenados de acuerdo al nombre de la provincia, el dato corresponderá a los años de los pacientes.
- -casos_cui: muestra de casos en cuidados intensivos ordenados por fecha. En caso de que se haya insertado un parámetro, se presenta solo las fechas mayores a esta.

La nomenclatura y implementación de argumentos se basaron en la explicación realizada del tema en la clase del 28/10. Se utiliza bibliotecas iostream y string (lectura de argumentos).

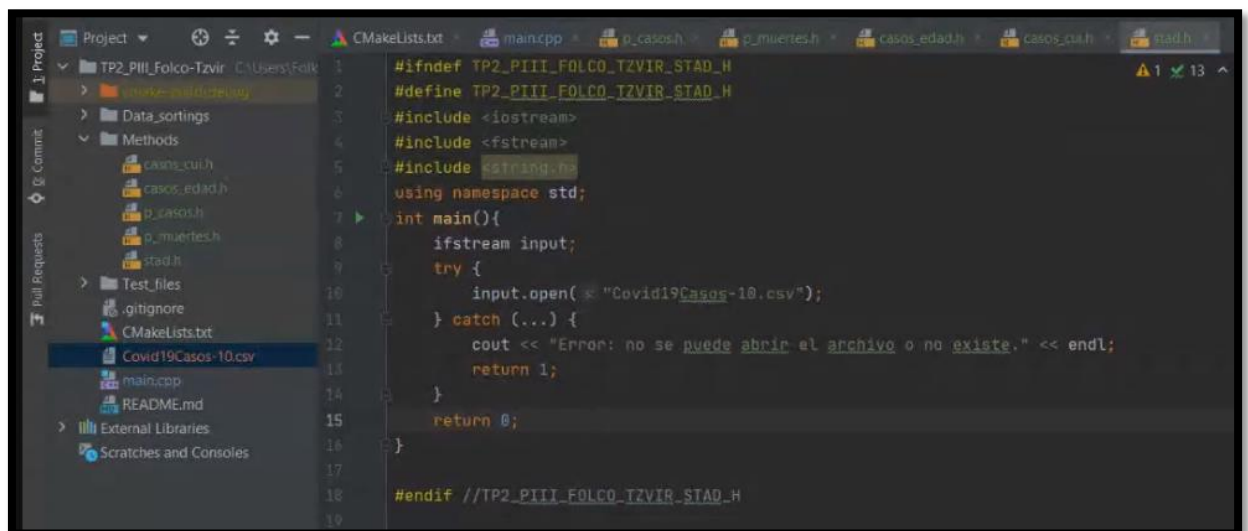


Luego, se realiza el agregado de un nuevo directorio con los métodos a utilizar para mejor organización.

Lo siguiente es la realización de los métodos **“stad.h”**:



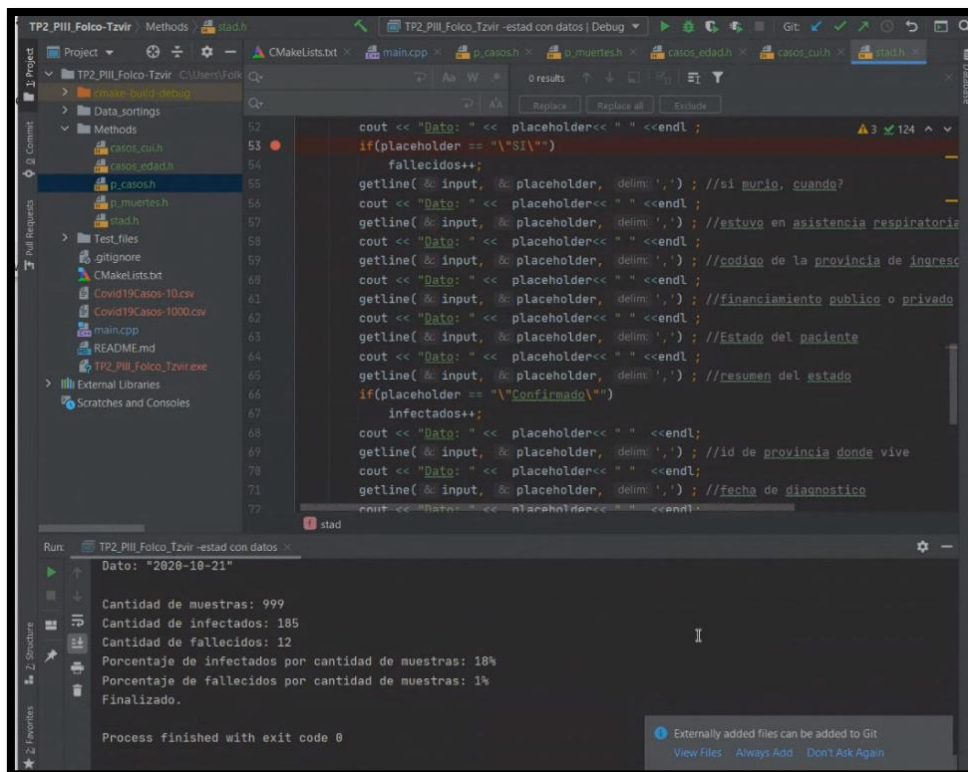
El paso realizado luego es el agregado de código para la apertura y lectura del archivo en .CSV. a través del uso de la librería fstream.



La lectura de los datos consiste en la utilización de getline de la biblioteca string, de modo que lea la información presente del archivo.

La intención de este algoritmo es la presentación de la información en forma ordenada, y luego los datos relacionados a lo estadístico.

Los principales inconvenientes fueron los problemas relacionados al espacio entre cada categoría (solucionado con agregado de comas y endl) y la muestra de los datos “**cantidad total de muestras**”, “**cantidad total de infectados**”, “**cantidad de fallecidos**”, “**% de infectado por muestras**” y “**% de fallecidos por infectados**” (resuelto con cálculos).



Tarea: Se encuentra pendiente “Cantidad de infectados por rango etario (rango de 10 años)” y “Cantidad de muertes por rango etario (rango de 10 años)”, documentar, ver el impacto en performance y la evaluación de optimización de algoritmo.

16/11: CONSTRUCCIÓN DE LA CLASE Y REALIZACIÓN DE LOS DEMÁS ARGUMENTOS

Se lleva a cabo en primer lugar la realización de la clase “Paciente.h” para continuar con el proyecto.

```
int id;
char genero;
int edad; //en meses
std::string pais;
std::string provincia;
std::string departamento;
std::string provinciaC;
Fecha sintomas;
Fecha inicio_caso;
int semana_inicio;
Fecha internacion;
bool CUI;
Fecha CUIF;
bool fallecido;
Fecha fallecimiento;
bool asistenciareso;
int idprovinciacarga;
std::string financiamiento;
std::string clasificacion;
std::string resumen;
int idprovinciaries;
Fecha diagnostico;
int iddepartamentores;
```

El objetivo diario es el boceto de las funciones, armando los argumentos, teniendo el cuerpo de las mismas desarrollado. Finalizado ese proceso, se realizará el pulido y el perfeccionamiento de las mismas.

Continuando con la clase “stad.h”, se decide la implementación de la estructura de datos de árbol binario, con el criterio de que la misma es la mejor herramienta para el manejo y ordenamiento de los datos a asignar, siendo los “cantidad de infectados por rango etario (rango de 10 años)” y “cantidad de muertes por rango etario (rango de 10 años)”. Se incorpora el nodo (“TreenodeStad.h”) y clase.h (“BinarytreeStad.h”). Sin embargo, la ejecución de la misma se llevará a cabo en otro momento. También se agregan los métodos “fecha.h” y “fecha.cpp” para la interpretación de las fechas de los archivos .CSV.

```

1  #ifndef UB6_ARBOL_ARBOL_BINARYTREE_H_
2  #define UB6_ARBOL_ARBOL_BINARYTREE_H_
3
4  #include "TreenodeStad.h"
5  #include <iostream>
6  #include <string>
7
8  // CRUD
9  %
10 %
11 %
12 %
13 %
14 %
15 %
16 %
17 %
18 %
19 %
20 %
21 %
22 %
23 %
24 %
25 %
26 %
27 %
28 %
29 %
30 %
31 %
32 %
33 %
34 %
35 %
36 %
37 %
38 %
39 %
40 %
41 %
42 %
43 %
44 %
45 %
46 %
47 %
48 %
49 %
50 %
51 %
52 %
53 %
54 %
55 %
56 %
57 %
58 %
59 %
60 %
61 %
62 %
63 %
64 %
65 %
66 %
67 %
68 %
69 %
70 %
71 %
72 %
73 %
74 %
75 %
76 %
77 %
78 %
79 %
80 %
81 %
82 %
83 %
84 %
85 %
86 %
87 %
88 %
89 %
90 %
91 %
92 %
93 %
94 %
95 %
96 %
97 %
98 %
99 %
100 %

```

A continuación, se borra la primera línea de los archivos .CSV para evitar que el programa tome ese fragmento como error y se ejecute los comandos correctamente.

```

1  "1000000","M","03","Años","Argentina","CABA","SIN ESPECIFICAR","Buenos Aires","2020-06-01","23","NO","NO","NO","06","Privado","Caso descartado"
2  "1000002","M","21","Años","Argentina","Buenos Aires","La Matanza","Buenos Aires","2020-06-01","23","NO","NO","NO","06","Publico","Caso descartado"
3  "1000003","F","40","Años","Argentina","Córdoba","Capital","Córdoba","2020-05-24","2020-06-01","23","NO","NO","NO","14","Privado","Caso descartado"
4  "1000005","F","58","Años","Argentina","Mendoza","Las Heras","Mendoza","2020-06-01","23","NO","NO","NO","06","Publico","Caso descartado"
5  "1000006","M","28","Años","Argentina","Buenos Aires","Malvinas Argentinas","Buenos Aires","2020-05-30","2020-06-01","23","NO","NO","NO","06","Publico","Caso descartado"
6  "1000007","M","26","Años","Argentina","Formosa","Patino","Formosa","2020-06-01","23","NO","NO","NO","14","Publico","Caso confirmado por labor"
7  "1000008","F","49","Años","Argentina","CABA","SIN ESPECIFICAR","CABA","2020-06-01","2020-06-01","23","NO","NO","NO","02","Privado","Caso descartado"
8  "1000009","M","73","Años","Argentina","Buenos Aires","Esteban Echeverría","Buenos Aires","2020-05-31","2020-06-01","23","NO","NO","NO","06","Publico","Caso descartado"
9  "1000010","M","7","Años","Argentina","CABA","COMUNA 07","CABA","2020-06-01","23","NO","NO","NO","02","Publico","Caso confirmado por labor"
10 "1000011","M","42","Años","Argentina","Santa Fe","Rosario","Santa Fe","2020-06-01","23","NO","NO","NO","02","Publico","Caso descartado"
11 "1000012","M","44","Años","Argentina","CABA","SIN ESPECIFICAR","CABA","2020-05-19","2020-06-01","23","2020-05-31","NO","NO","NO","02","Privado","Caso descartado"
12 "1000013","F","43","Años","Argentina","Buenos Aires","Tigre","Buenos Aires","2020-06-01","23","NO","NO","NO","06","Privado","Caso descartado"
13 "1000014","M","40","Años","Argentina","Santa Fe","Rosario","Santa Fe","2020-06-01","23","NO","NO","NO","02","Publico","Caso descartado"
14 "1000015","F","29","Años","Argentina","CABA","COMUNA 07","Buenos Aires","2020-05-19","2020-06-01","23","NO","NO","NO","06","Privado","Caso descartado"
15 "1000016","F","37","Años","Argentina","Buenos Aires","Tres de Febrero","Buenos Aires","2020-05-29","2020-06-01","23","2020-05-30","NO","NO","NO","06","Privado","Caso descartado"

```

Con “stad.h”, una de las dificultades es hacer que el programa al momento de la lectura de la edad hiciera que la misma se presentara en años (cuando en los .CSV original se encuentra personas con años y meses). Para ello se implementa los getline de la función string y el agregado de un condicional que indica que si hay algún integrante con meses, se pasara la edad a 0 años.

```

string placeholder;
int total=0,edad=0,infectados=0,fallecidos=0;
BinaryTree<unsigned int> rango;
// Cantidad de infectados por rango etario (rango de 10 años)
// Cantidad de muertes por rango etario (rango de 10 años)
while(getline(&input, &placeholder, delim ',')){
    total++;
    cout << "Caso " << total << " " << placeholder << " " << endl; //10

    getline(&input, &placeholder, delim ',') ; //el Date no importa

    getline(&input, &placeholder, delim ',') ; //edad
    cout << "Edad: " << placeholder << " " << endl ;
    stringstream temp(placeholder);
    temp >> edad;
    getline(&input, &placeholder, delim ',') ; //si la edad son años o meses, ignorar
    cout << "Edad en meses?: " << placeholder << " " << endl ;

    getline(&input, &placeholder, delim ',') ; //pais donde vive

    getline(&input, &placeholder, delim ',') ; //proximidad donde vive
    cout << "Provincia: " << placeholder << " " << endl;

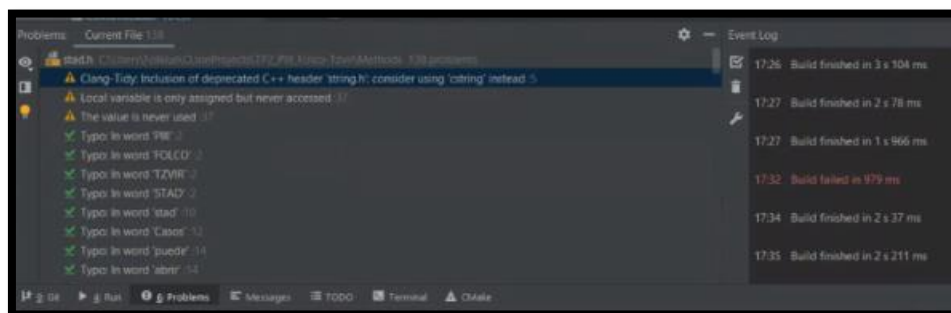
```

```

if(placeholder == "\"Meses\""){
    edad=0;
}

```

Se han presentado algunas complicaciones, teniendo algunos errores como el presentado debajo. Esta dificultad luego será corregida.



Refiriéndose a la utilización de estructuras de datos, la decisión conjunta fue que se leyera los argumentos con listas, con el fundamento de que posee medidas de Big O notation favorables (siendo $O(1)$ en el mejor de los casos), por su capacidad de ordenamiento y por el hecho de no se requiere la necesidad de extraer datos.

Se aplica esta estructura contenedora en “p_casos.h”.


```

        if (placeholder == "\\Confirmado\\") {
            infectados++;
            estado temp;
            temp.cantidad++;
            temp.nombre=provincia;
            provincias().push_front(temp);
        }

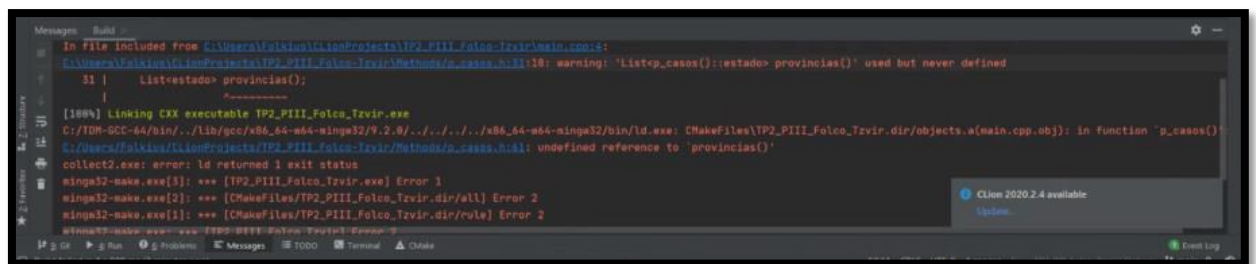
        cout << "Estado: " << placeholder << " " << endl;
        getline(& input, & placeholder, delim); //id de provincia donde vive
        cout << "ID provincial: " << placeholder << " " << endl;
        getline(& input, & placeholder, delim); //fecha de diagnostico
        getline(& input, & placeholder, delim); //id de departamento donde vive
        getline(& input, & placeholder); //ultima actualizacion
        cout << endl;

    }

    return 0;
}

```

Presentándose el inconveniente de que el programa no le parece adecuada la estructura.



Realizando la corrección pedida por el programa, se logra la compilación del programa.

```

74         getline(& input, & placeholder, delim); //Estado del paciente
75         getline(& input, & placeholder, delim); //resumen del estado
76         if (placeholder == "\\Confirmado\\") {
77             infectados++;
78             estado temp(provincia, cantidad);
79             provincias().push_front(temp);
80         }
81         cout << "Estado: " << placeholder << " " << endl;
82         getline(& input, & placeholder, delim); //id de provincia donde vive
83         cout << "ID provincial: " << placeholder << " " << endl;
84         getline(& input, & placeholder, delim); //fecha de diagnostico
85         getline(& input, & placeholder, delim); //id de departamento donde vive
86         getline(& input, & placeholder); //ultima actualizacion
87         cout << endl;
88     }
89     return 0;
90 }
91 #endif //TP2_PIII_FOLCO_TZVIR_P_CASOS_H

```

```

TP2_PIII_Folco_Tzvir-g_casos no se
Estado: "UNSCA"
ID provincial: "96"

Caso 9 "1888018"
Provincia: "CABA"
Estado: "Confirmado"
ID provincial: "82"

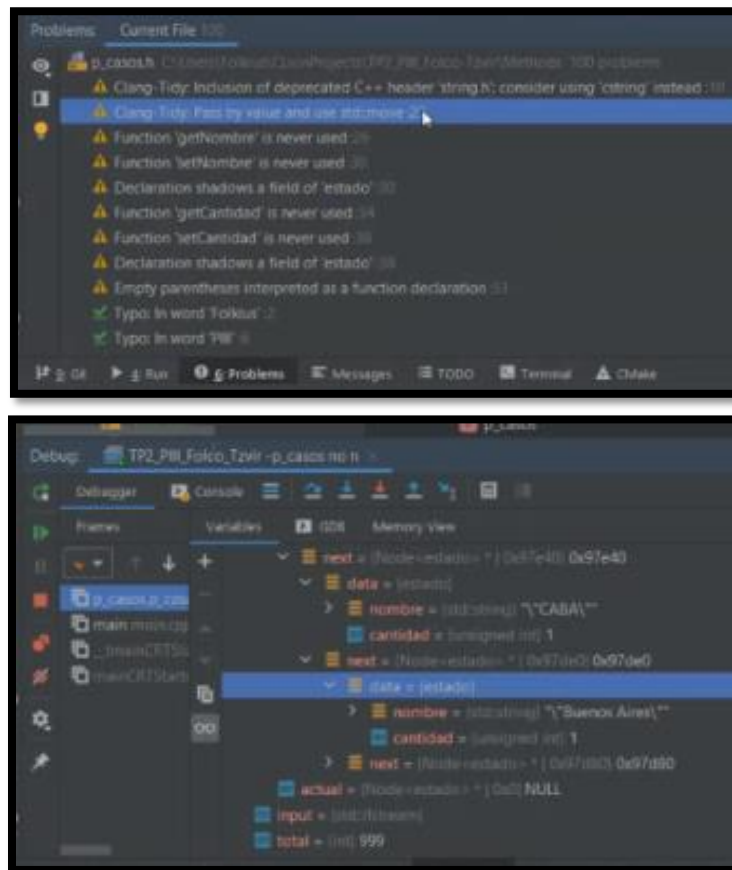
Finalizado.

Process finished with exit code 0

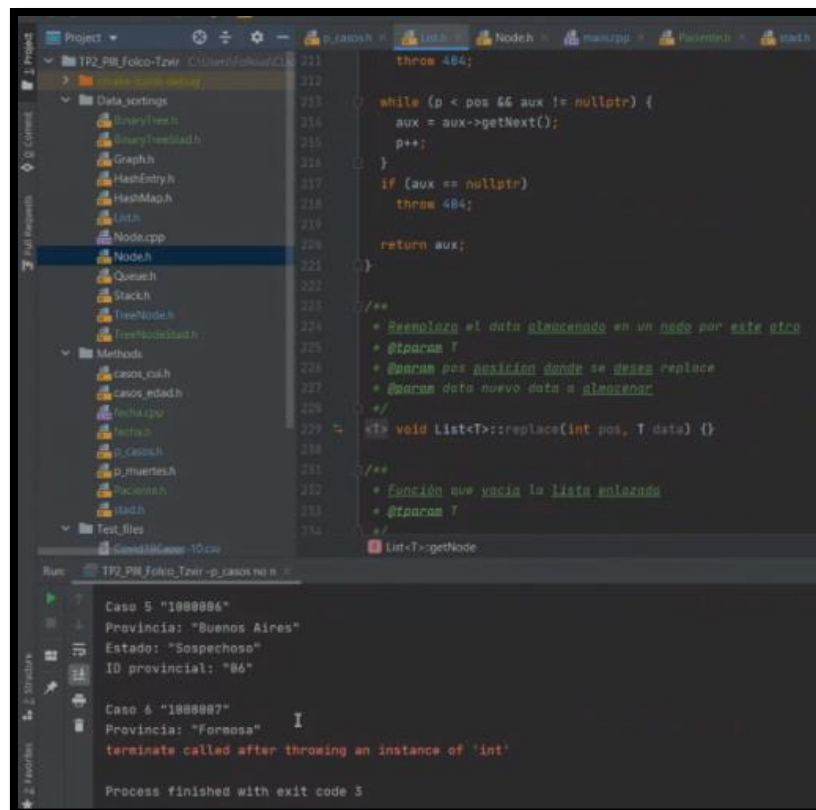
```

Siguiendo con “p_casos.h”, ahora se busca aplicar un algoritmo que permita la contabilización de la cantidad de infectados por provincia.

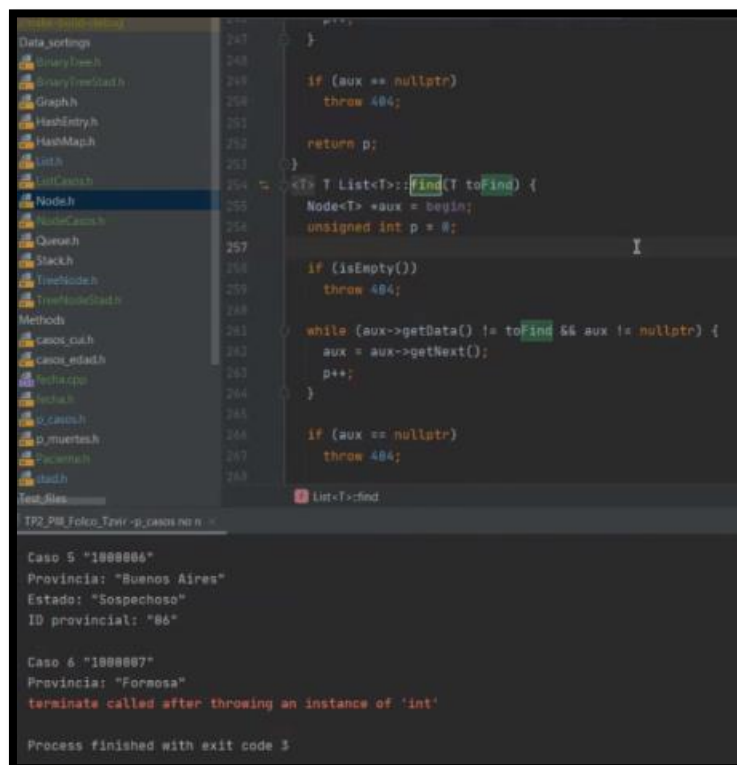
Intentando implementar este algoritmo, se han presentado complicaciones como la repetición de las provincias, y algunas advertencias de funciones no utilizadas. Se busca solucionarlo con un algoritmo de búsqueda, de tal forma que este error no suceda.



Sin embargo, no hay éxito y el problema permanece.



Como último paso, se procede a realizar una copia de las estructuras de datos, de manera tal que se pueda hacer cambios que formen parte de este programa.

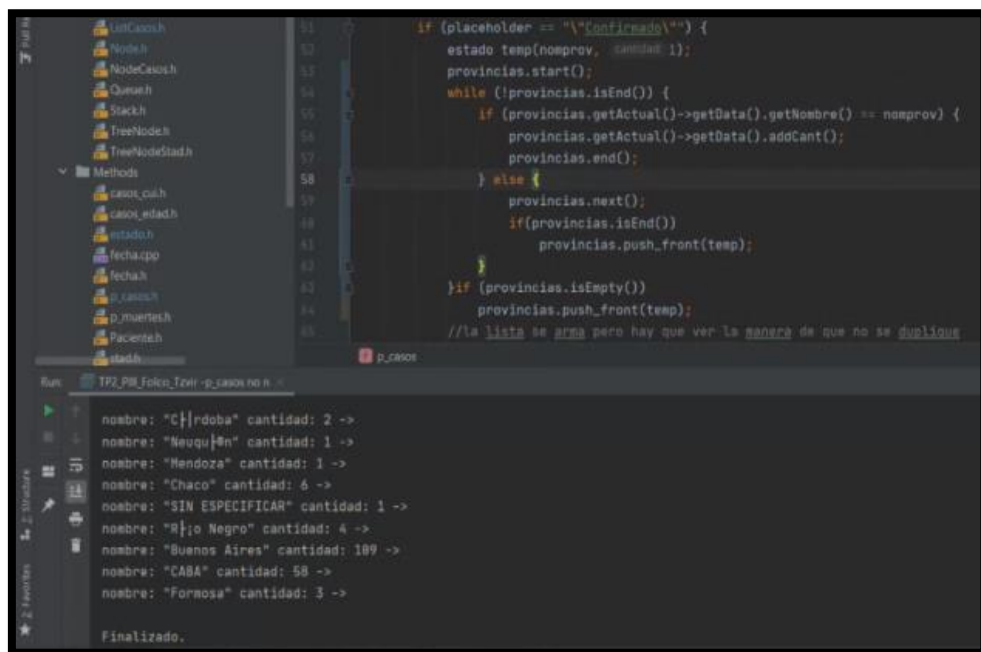


Tarea: se debe solucionar el contado de provincias, buscar algoritmos de ordenamiento para los argumentos que lo requieran, desarrollar árbol balanceado, documentar, y analizar refinamiento del programa y performance.

17/11: SOLUCIÓN DE PROBLEMAS Y ARMADO DE ARGUMENTO

"CASOS EDAD.H"

Se lleva a cabo la solución de cómo contar las provincias. Se utiliza el método **"find"** perteneciente a la lista, y se aplica un while con condiciones (si apareció la provincia en la lista, se la agrega a un contador. En caso contrario, se agrega una nueva provincia y se inicializa cuenta). También se le realiza una clase particular para este argumento (**"estado.h"**).



```
11 if (placeholder == "\\Confirmado\\") {
12     estado temp(nomprov, cantidad);
13     provincias.start();
14     while (!provincias.isEnd()) {
15         if (provincias.getActual()->getData().getNombre() == nomprov) {
16             provincias.getActual()->getData().addCant();
17             provincias.end();
18         } else {
19             provincias.next();
20             if (provincias.isEnd())
21                 provincias.push_front(temp);
22         }
23     } if (provincias.isEmpty())
24         provincias.push_front(temp);
25     //la lista se arma pero hay que ver la manera de que no se duplique
```

Run: TP2_P01_Folios_T2H1 - p_casos no n. ->

```
nombre: "Chubut" cantidad: 2 ->
nombre: "Neuquén" cantidad: 1 ->
nombre: "Mendoza" cantidad: 1 ->
nombre: "Chaco" cantidad: 6 ->
nombre: "SIN ESPECIFICAR" cantidad: 1 ->
nombre: "Río Negro" cantidad: 4 ->
nombre: "Buenos Aires" cantidad: 109 ->
nombre: "CABA" cantidad: 58 ->
nombre: "Formosa" cantidad: 3 ->
Finalizado.
```

En segundo paso, se procede a continuar con **"casos_edad.h"**. Se agrega los getline y métodos de lista correspondiente. Ahora se analiza el método para ver la edad en años.

Surge la problemática de haber lectura de datos, pero no de mostrar por separado las edades de los archivos .CSV.

```

63     getline(&input, &placeholder, delim, &); //si murio, cuando?
64     getline(&input, &placeholder, delim, &); //estuvo en asistencia respiratoria?
65     getline(&input, &placeholder, delim, &); //codigo de la provincia de ingreso
66     getline(&input, &placeholder, delim, &); //financiamiento publico o privado
67     getline(&input, &placeholder, delim, &); //Estado del paciente
68     getline(&input, &placeholder, delim, &); //resumen del estado
69     getline(&input, &placeholder, delim, &); //id de provincia donde vive
70     getline(&input, &placeholder, delim, &); //fecha de diagnostico
71     getline(&input, &placeholder, delim, &); //id de departamento donde vive
72     stringstream temp(placeholder);
73     temp >> id;
74     getline(&input, &placeholder); //ultima actualizacion
75     cout << endl;
76 }
77 return 0;
78 }
79 #endif //TP2_PIII_FOLCO_TZVIR_CASOS_EDAD_H

```

Run: TP2_PIII_FOLCO_TZVIR_CASOS_EDAD_H
 Estado: "CABA" cantidad: 4 ->
 ID provincial: "14"
 nombre: "Corrientes" cantidad: 1 ->
 nombre: "Río Negro" cantidad: 1 ->
 nombre: "CABA" cantidad: 4 ->
 nombre: "Buenos Aires" cantidad: 6 ->
 Finalizado.
 Process finished with exit code 0

Para solucionarlo, se realiza nuevo constructor en clase **"Paciente.h"** y agregado de nuevas variables, algunas cambiando tipo de parámetro a `std::string`.

```

Paciente temp(id, genero, edad, nompais, nomprov, nomdep, nomprove, iniciosint, medicoc, semmedicoc, fechaint, id);
Paciente(int id, char genero, int edad, const std::string &pais, const std::string &provincia,
const std::string &departamento, const std::string &provinciaC, const Fecha &sintomas,
const Fecha &inicioCaso, int semanaInicio, const Fecha &internacion, bool cui, const Fecha &cuif,
bool fallecido, const Fecha &fallecimiento, bool asistenciaresp, int idprovinciacarga,
const std::string &financiamiento, const std::string &clasificacion, const std::string &resumen,
int idprovinciaries, const Fecha &diagnostico, int iddepartamentores, const Fecha &actualizacion) : id(id),
genero(genero), edad(edad), pais(pais), provincia(provincia), departamento(departamento),
provinciaC(provinciaC), sintomas(sintomas), inicio_caso(inicioCaso), semana_inicio(semanaInicio),
internacion(internacion), CUI(cui), CUIF(cuif), fallecido(fallecido), fallecimiento(fallecimiento),
asistenciaresp(asistenciaresp), idprovinciacarga(idprovinciacarga), financiamiento(financiamiento),
clasificacion(clasificacion), resumen(resumen), idprovinciaries(idprovinciaries), diagnostico(diagnostico),
iddepartamentores(iddepartamentores), actualizacion(actualizacion) {}

return 0;
}
#endif //TP2_PIII_FOLCO_TZVIR_CASOS_EDAD_H

```

```
#include <...>

class Paciente {
private:
    int id;
    char genero;
    int edad; //en meses
    std::string pais;
    std::string provincia;
    std::string departamento;
    std::string provinciaC;
    Fecha sintomas;
    Fecha inicio_caso;
    int semana_inicio;
    std::string internacion;
    std::string CUI;
    Fecha CUIF;
    bool fallecido;
    Fecha fallecimiento;
    bool asistenciaresp;
public:
```

```
16     int edad; //en meses
17     std::string pais;
18     std::string provincia;
19     std::string departamento;
20     std::string provinciaC;
21     std::string sintomas;
22     std::string inicio_caso;
23     int semana_inicio;
24     std::string internacion;
25     std::string CUI;
26     std::string CUIF;
27     std::string fallecido;
28     std::string fallecimiento;
29     std::string asistenciaresp;
30     int idprovinciacarga;
31     std::string financiamiento;
32     std::string clasificacion;
33     std::string resumen;
34     int idprovinciases;
35     std::string diagnostico;
36     int iddepartamentases;
37     std::string actualizacion;
38 public:
39     Paciente() {}
```

Paciente Paciente

Problems: Current File 237

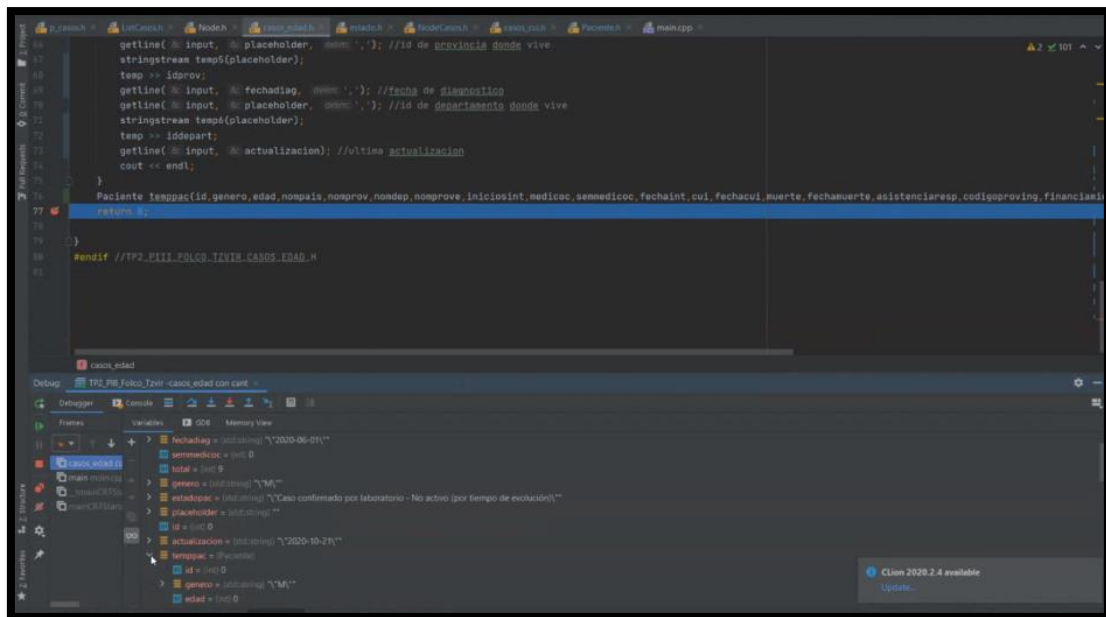
Paciente.h C:\Users\Felipe\OneDrive\Projects\TF2_PBL_Force-Tizen\Methods_237\problems

- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'char' :87
- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'const Fecha' :143
- No viable overloaded '=' :145
- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'const Fecha' :153
- No viable overloaded '=' :157
- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'const Fecha' :168
- No viable overloaded '=' :173
- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'bool' :177
- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'const Fecha' :180
- No viable overloaded '=' :185
- No viable conversion from returned value of type 'const std::string' (aka 'const basic_string<char>') to function return type 'bool' :193

Go Run Problems Messages Debug TODO Terminal Make

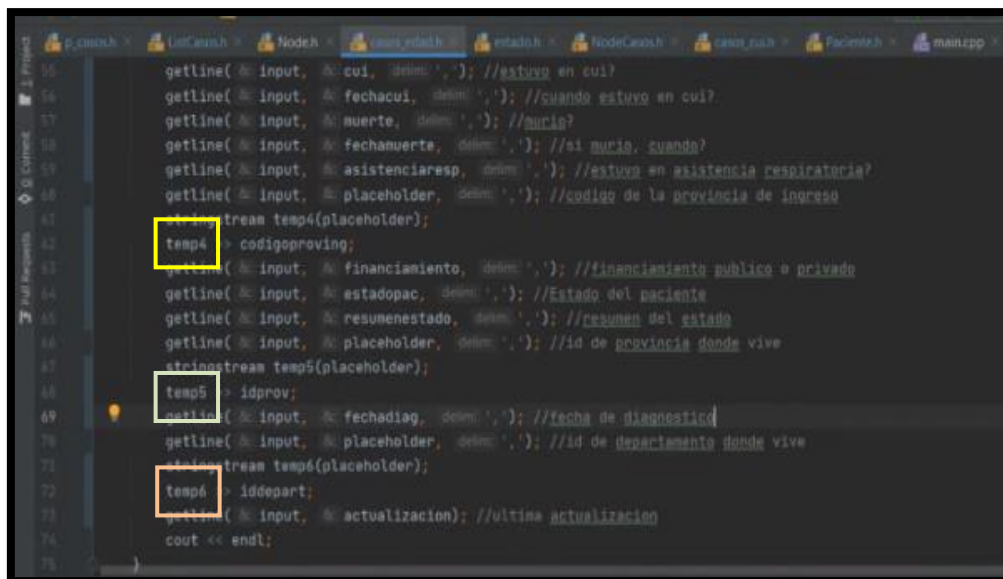
Surge un pequeño problema de que hay ciertas conversiones que no toma en cuenta, siendo solucionadas con el borrado y vuelta de generación de setters y getters de la clase.

Volviendo a “casos_edad.h”, se presenta otro problema: no da de manera correcta los números de los archivos .CSV, a pesar de que hay compilación.



Se intentaron variadas formas de solucionarlo:

- 1- Cambio de nombres de stringstream Temp



- 2- Agregado de ostream:

3- Cambios de variable:

```

string muerte, fechamuerte, asistenciaresp, financiamiento, estadopac, resumenestado, fechadiag,
int total = 0, edad=0, id=0, semedicoc=0, codigoprovinc=0, idprov=0, iddepart=0, matching=0;
List<Paciente> delaedad;
while (getline(& input, & placeholder, delim: ',')) {
    total++;
    stringstream temp(placeholder);
    temp >> id;
    cout << "Caso " << total << " " << placeholder << " " << endl; //ID
    getline(& input, & genero, delim: ','); //Genero
    getline(& input, & placeholder, delim: ','); //edad
    cout << "Edad: " << placeholder << " " << endl;
    stringstream temp2(placeholder);
    edad=std::stoi(placeholder);
    //temp2 >> edad;
    getline(& input, & placeholder, delim: ','); //si la edad son años o meses
    if (placeholder == "\\Meses\\") {
        // solucionar el stringstream que no toma bien
    }
    getline(& input, & nompais, delim: ','); //pais donde vive
    getline(& input, & nomprov, delim: ','); //provincia donde vive
    getline(& input, & ncodep, delim: ','); //departamento donde vive
}

```

Sin embargo, el inconveniente sigue presentándose.

Tarea: solucionar lectura correcta de la edad, buscar algoritmos de ordenamiento para los argumentos que lo requieran, desarrollar árbol balanceado, documentar, y analizar refinamiento del programa y performance.

**18/11: SOLUCIÓN DE PROBLEMAS Y DESARROLLO DE “CASOS EDAD.H” Y
“P CASOS.H”**

Para solucionar la lectura correcta de la edad en “**casos_edad.h**”, de mantuvo el stoi, y se agregaron antes y al final de la línea de edad para solamente captar las comillas.


```

24 cout << "Leyendo archivo..." << endl;
25
26 string placeholder, genero, nonprov, nonpais, nondep, nonprove, inicioint, medicos, fechaint, cui, fechasui;
27 string muerte, fechamuerte, asistenciaresp, financiamiento, estadopac, resumenestado, fechadiag, actualizaci;
28
29 int total = 0, edad=0, id=0, semmedicoc=0, codigoprevieng=0, idprov=0, iddepart=0;
30 List<Paciente> delaedad;
31 while (getline( & input, & placeholder, & delimi ',' )) {
32     total++;
33     stringstream temp(placeholder);
34     temp >> id;
35     cout << "Caso " << total << " " << placeholder << " " << endl; //ID
36     getline( & input, & genero, & delimi ',' ); //Genero
37     getline( & input, & placeholder, & delimi ',' ); //Cantar inicio de comillas
38     getline( & input, & placeholder, & delimi ',' ); //edad
39     cout << "Edad: " << placeholder << " " << endl;
40
41     try{
42         edad=std::stoi(placeholder); //¿Esta la edad puesta?
43     }catch(...){
44         edad=0; //Si no tiene edad se deja vacio
45     }
46
47     getline( & input, & placeholder, & delimi ',' ); // Cantar fin de comillas
48     getline( & input, & placeholder, & delimi ',' ); //si la edad son años o meses
49     if (placeholder == "Meses") {
50         edad = 0;
51     }
52 }

```

Run: TP2_PII_Folco_Tzvir -casos_edad con cant -

```

C:\Users\Folkius\CLionProjects\TP2_PII_Folco_Tzvir\cmake-build-debug\TP2_PII_Folco_Tzvir.exe -casos_edad 69 Covid19Casos-18.csv
Realizando analisis por edad... (con edad de 69 años)
Leyendo archivo...
Caso 1 "1800000"
Edad: 53
Caso 2 "1800002"
Edad: 21
Caso 3 "1800003"
Edad: 40

```

Luego, se lleva a cabo la corrección de los casos y agregado de espacios en blanco para separación.

```

101 try{
102     idprov=std::stoi(placeholder);
103 }catch(...){
104     idprov=0;
105 }
106
107 getline( & input, & placeholder, & delimi ',' );
108
109 getline( & input, & fechadiag, & delimi ',' ); //fecha de diagnostico
110
111 getline( & input, & placeholder, & delimi ',' );
112
113 getline( & input, & placeholder, & delimi ',' ); //id de departamento donde vive
114
115 try{
116     iddepart=std::stoi(placeholder); //¿Esta la edad puesta?
117 }catch(...){
118     iddepart=0;
119 }
120
121
122 getline( & input, & placeholder, & delimi ',' );
123
124 getline( & input, & actualizacion); //ultima actualizacion
125 cout << endl;
126 if (edad >= 0){
127     Paciente temppac(id, genero, edad, nonpais, nonprov, nondep, nonprove, inicioint, medicos, semmedicoc,
128     delaedad.push_front(temppac);
129 }

```

Run: TP2_PII_Folco_Tzvir -casos_edad con cant -

```

C:\Users\Folkius\CLionProjects\TP2_PII_Folco_Tzvir\cmake-build-debug\TP2_PII_Folco_Tzvir.exe -casos_edad 69 Covid19Casos-18.csv
Realizando analisis por edad... (con edad de 69 años)
Leyendo archivo...
Caso 1 "1800000"
Edad: 53
Caso 2 "1800002"
Edad: 21
Caso 3 "1800003"
Edad: 40

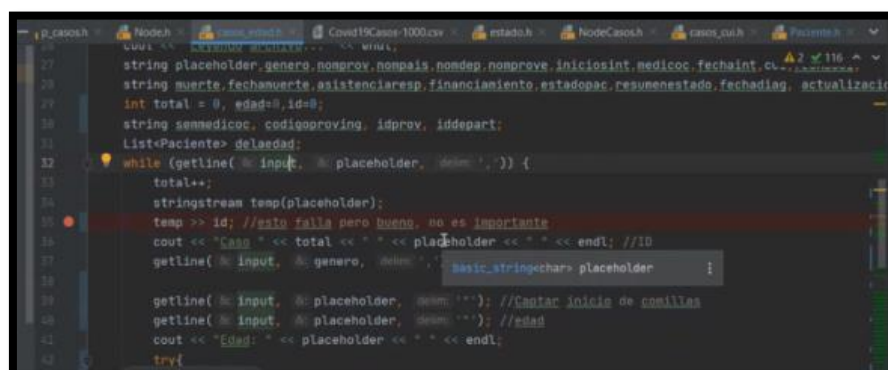
```

Otro paso hecho es la inserción de meses y días en la clase “Paciente.h”, además de cambiar el constructor.

```
std::string provincia;
std::string departamento;
std::string provinciaC;
std::string sintomas;
std::string inicio_caso;
int semana_inicio;
std::string semana_iniciostr;
std::string internacion;
std::string CUI;
std::string CUIF;
int mesCUIF;
int diaCUIF;
std::string fallecido;
std::string fallecimiento;
std::string asistenciaresp;
int idprovinciacarga;
std::string idprovinciacargastr;
std::string financiamiento;
std::string clasificacion;
std::string resumen;
int idprovinciaresta;
std::string idprovinciarestastr;
std::string diagnostico;
```

Retomando el argumento “casos_edad.h”, se procede a la extracción de comillas, debido a que el programa no funcionaba bien con las mismas (no tomaba en cuenta ciertos casos de edades por problemas de lectura).

Un problema generado es que el id no funciona, solo lo lee.



```
27 string placeholder.genero,nomprov,nompais,nomdep,nomprove,iniciostr,medicoc,fechastr,cl...
28 string muerte,fechamuerte,asistenciaresp,financiamiento,estadopac,resumenestado,fechadiag,actualizaci...
29 int total = 0, edad=0, id=0;
30 string senmedicoc, codigoprovinc, idprov, iddepart;
31 List<Paciente> delaedad;
32 while (getline( input, placeholder, '\n' )) {
33     total++;
34     stringstream temp(placeholder);
35     temp >> id; //esto falla pero bueno, no es importante
36     cout << "Caso " << total << " " << placeholder << " " << endl; //ID
37     getline( input, placeholder, '\n' );
38     getline( input, placeholder, '\n' ); //Contar inicio de comillas
39     getline( input, placeholder, '\n' ); //edad
40     cout << "Edad: " << placeholder << " " << endl;
41 }
42 }
```

Como solución se realiza string de id para tener facilidad de lectura, se realiza también cambio de ofstream.

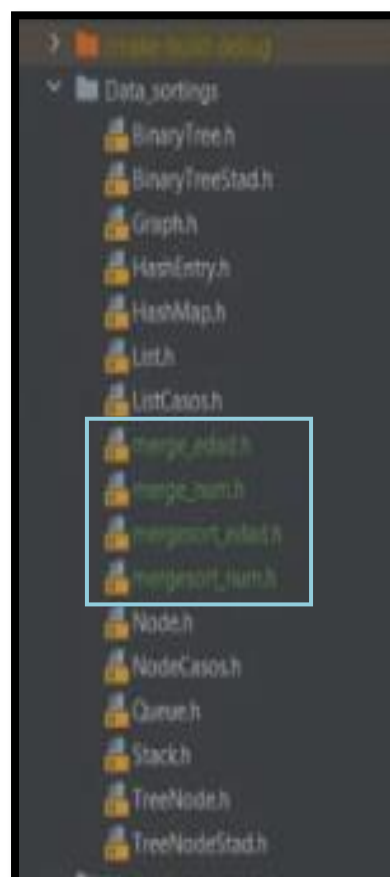
```

friend std::ostream &operator<<(std::ostream &os, const Paciente &paciente) {
    os << "id: " << paciente.id << " genero: " << paciente.genero << " edad: " << paciente.edad << " pais: "
    << paciente.pais << " provincia: " << paciente.provincia << " departamento: " << paciente.departamento
    << " provinciaC: " << paciente.provinciaC << " sintomas: " << paciente.sintomas << " inicio_caso: "
    << paciente.inicio_caso << " semana_inicio: " << paciente.semana_inicio << " internacion: "
    << paciente.internacion << " CUI: " << paciente.CUI << " CUIF: " << paciente.CUIF << " fallecido: "
    << paciente.fallecido << " fallecimiento: " << paciente.fallecimiento << " asistenciaresp: "
    << paciente.asistenciaresp << " idprovinciacarga: " << paciente.idprovinciacarga << " financiamiento: "
    << paciente.financiamiento << " clasificacion: " << paciente.clasificacion << " resumen: "
    << paciente.resumen << " idprovinciaries: " << paciente.idprovinciaries << " diagnostico: "
    << paciente.diagnostico << " iddepartamentores: " << paciente.iddepartamentores << " actualizacion: "
    << paciente.actualizacion;
    return os;
}

```

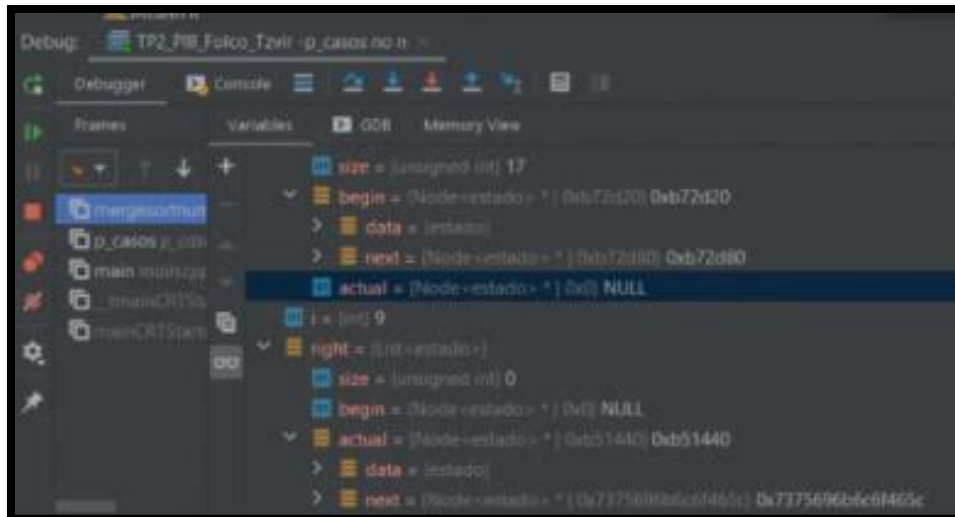
Con esos pasos ya se soluciona el problema y se puede continuar el proyecto.

Lo siguiente es la generación de los algoritmos de ordenamiento. Se decide el uso de mergesort por sus características, como su Big O de $n \log n$ (siendo constante sin importar si hay un mejor o peor caso). Este algoritmo se aplica en para “casos_edad.h” y “p_casos.h”.

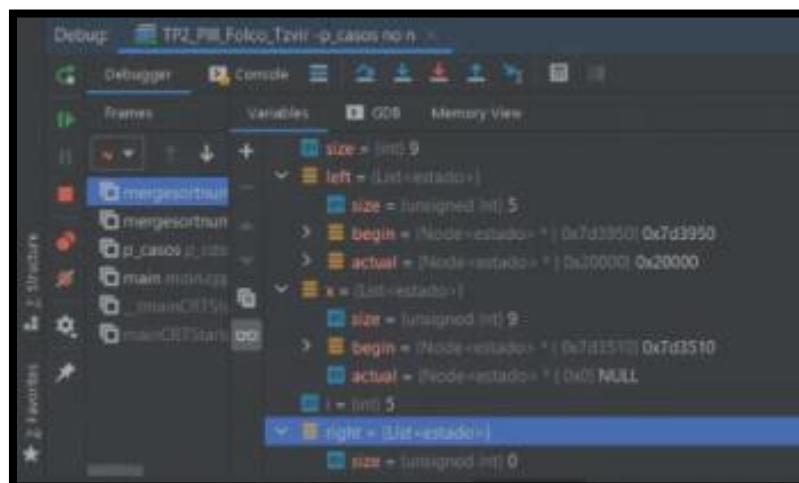


El algoritmo resulta tener ciertas complicaciones tales como:

- 1- Mostrar ciertos datos como NULL (se intenta solucionar el problema con un getnode público):



- 2- Duplicación del tamaño: al momento de no pasarle parámetros, el tamaño se duplica, por lo que entorpece la eficacia del método de ordenamiento. Si se coloca un tamaño fijo el programa funciona correctamente, tal como muestra la imagen siguiente.



Tarea: solucionar problema de duplicación de tamaño, desarrollar árbol balanceado, documentar, y analizar refinamiento del programa y performance.

19/11: AÑADIR DETALLES A MERGESORTNUM

Para el problema de duplicación de tamaño expuesto anteriormente, se añade una función en la lista que compruebe el mismo (**"checkSize"**).

```

List<estado> mergesortnum(List<estado> x) {
    x.checkSize();
    int size = x.getSize();
    // Base case. A list of zero or one elements is sorted, by defini
    if (size <= 1)
        return x;
}

```

Otro inconveniente que surge es que no se guardan los cambios hechos por el merge sort. Esto se debe a que el merge sort no modifica la lista nueva, sino que arma otra distinta.

Como solución, se genera una lista nueva, lográndose los resultados.

```
nombre: "Buenos Aires" cantidad: 109 ->
nombre: "CABA" cantidad: 58 ->
nombre: "Chaco" cantidad: 6 ->
nombre: "Río Negro" cantidad: 4 ->
nombre: "Formosa" cantidad: 3 ->
nombre: "Córdoba" cantidad: 2 ->
nombre: "SIN ESPECIFICAR" cantidad: 1 ->
nombre: "Neuquén" cantidad: 1 ->
nombre: "Mendoza" cantidad: 1 ->
```

Tarea: terminar con algoritmos de ordenamiento, darle como argumento el nombre del archivo de los casos (porque siempre se había dejado dentro de la función), documentar, y analizar refinamiento del programa y performance.

20/11: LECTURA DE RANGO ETARIO

En “stad.h”, debido a la dificultad de agregar el parámetro stringstream en un árbol balanceado binario, se realiza un cambio en el parámetro de lectura de edad, utilizando stoi, además de aplicar operaciones matemáticas para la obtención del rango, que será de utilidad al momento de realizar el árbol binario balanceado para ordenar y luego mostrar la información estadística “Cantidad de infectados por rango etario (rango de 10 años)” y “Cantidad de muertes por rango etario (rango de 10 años)”.

```
try {
    edad = std::stoi(placeholder); //¿Está la edad puesta?
} catch (...) {
    edad = 0; //Si no tiene edad se deja vacío
}
```

```
int mrango = edad / 10;
int minrango = mrango * 10;
int maxrango = mrango * 10 + 9;
rango_dato(minrango, maxrango, cantidad: 1);
cout << minrango << " " << maxrango << endl;
```

Tarea: terminar con algoritmos de ordenamiento y desarrollo de árbol balanceado, darle como argumento el nombre del archivo de los casos (porque siempre se había dejado dentro de la función), documentar, y analizar refinamiento del programa y performance.

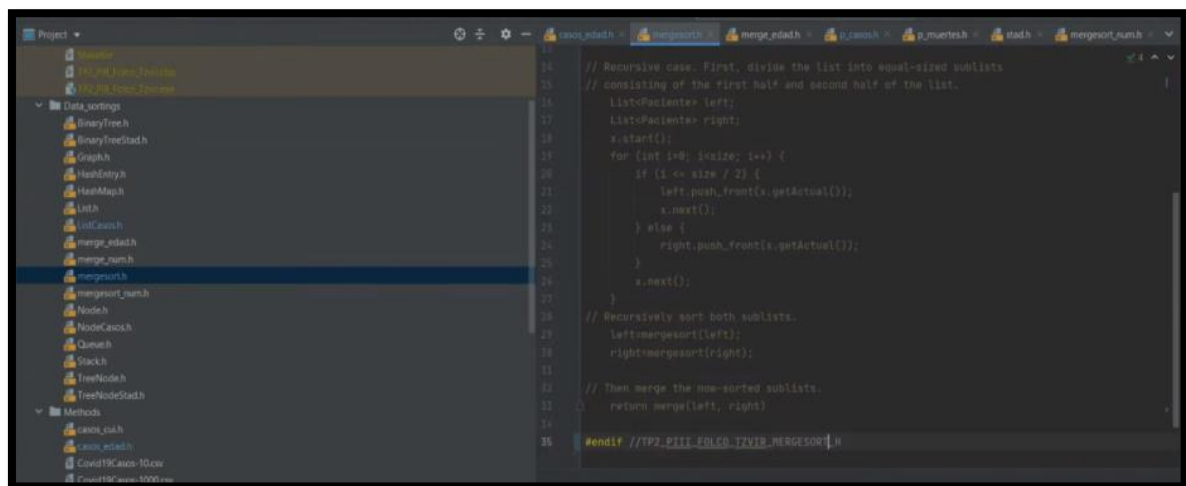
21/11: FINALIZACIÓN DE ARGUMENTOS

Se soluciona el problema de brindar como argumento el nombre del archivo de los casos.

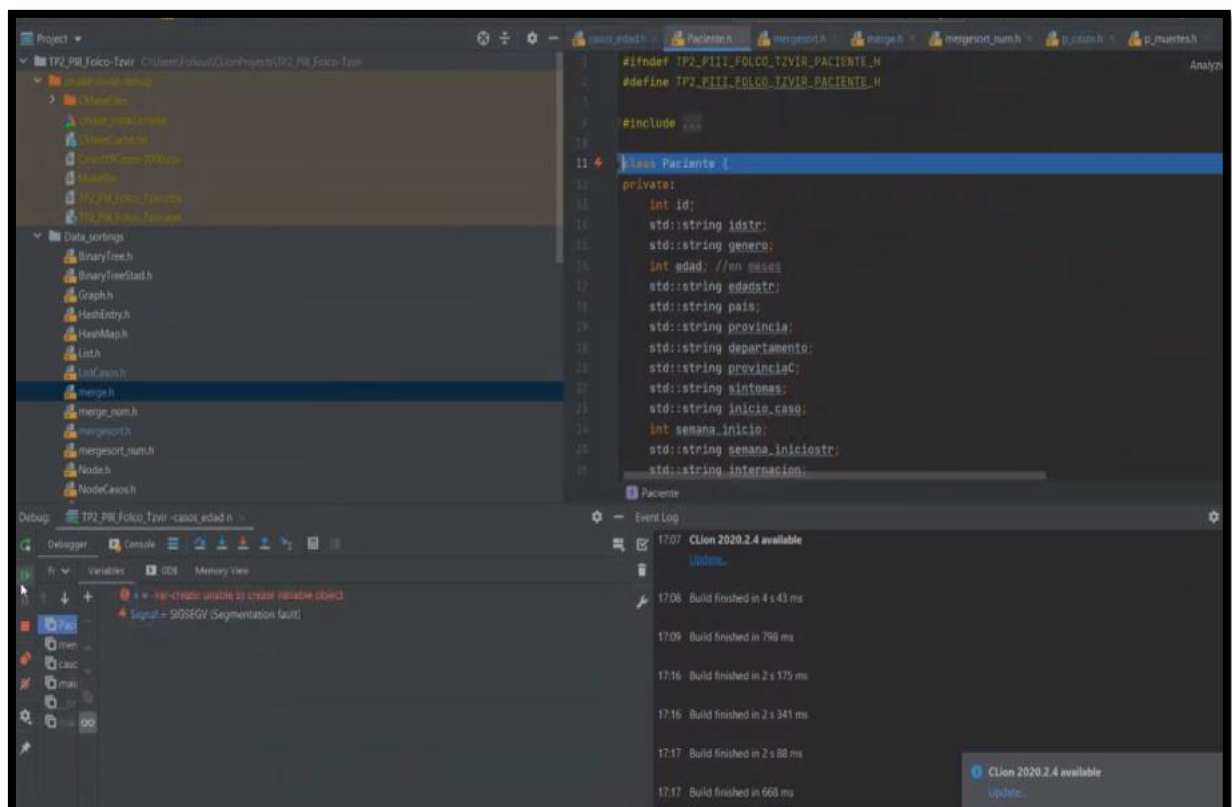
Luego, se procede a aplicar los algoritmos de ordenamiento pedidos en cada uno de los argumentos.

Se encuentran algunas dificultades que no permiten la lectura de mergesort:

1. A pesar de que el programa no ha sido comentado, se presenta como tal, por lo que al momento de la lectura no se aplica el algoritmo a donde la función es llamada.
Solución: cambio de nombre en las líneas “**indef**”, “**#define**” y “**#endif**”.



- 2- Sin embargo, el problema persiste. En principio se considera una falla de mergesort en el momento de armar la clase “**Paciente.h**”. Luego, se analiza por debugger y se concluye que la falla se debe a que sucede problemas con el tamaño.
Solución: se agrega la función de revisar tamaño (“**checkSize**”).



También se realiza una nueva lista. Habiendo hecho estas modificaciones, el algoritmo funciona correctamente.

```
List<Paciente> sorteado = mergesort(delaedad); //falta saber cómo ordenarlos por provincia
sorteado.print();
return 0;
```

Se aplica algoritmo de ordenamiento en “p_muertes.h”, “p_casos.h” y “casos_edad.h”, compilándose cada programa con éxito.

Para el momento de llevar a cabo de aplicarlo en “casos_cui.h”, se presenta el problema de que realiza el ordenamiento por provincias, no por fecha como indica el enunciado. La solución es cambiando de nombre la clase donde se encuentra el mergesort específico para fechas “mergefecha.h”.

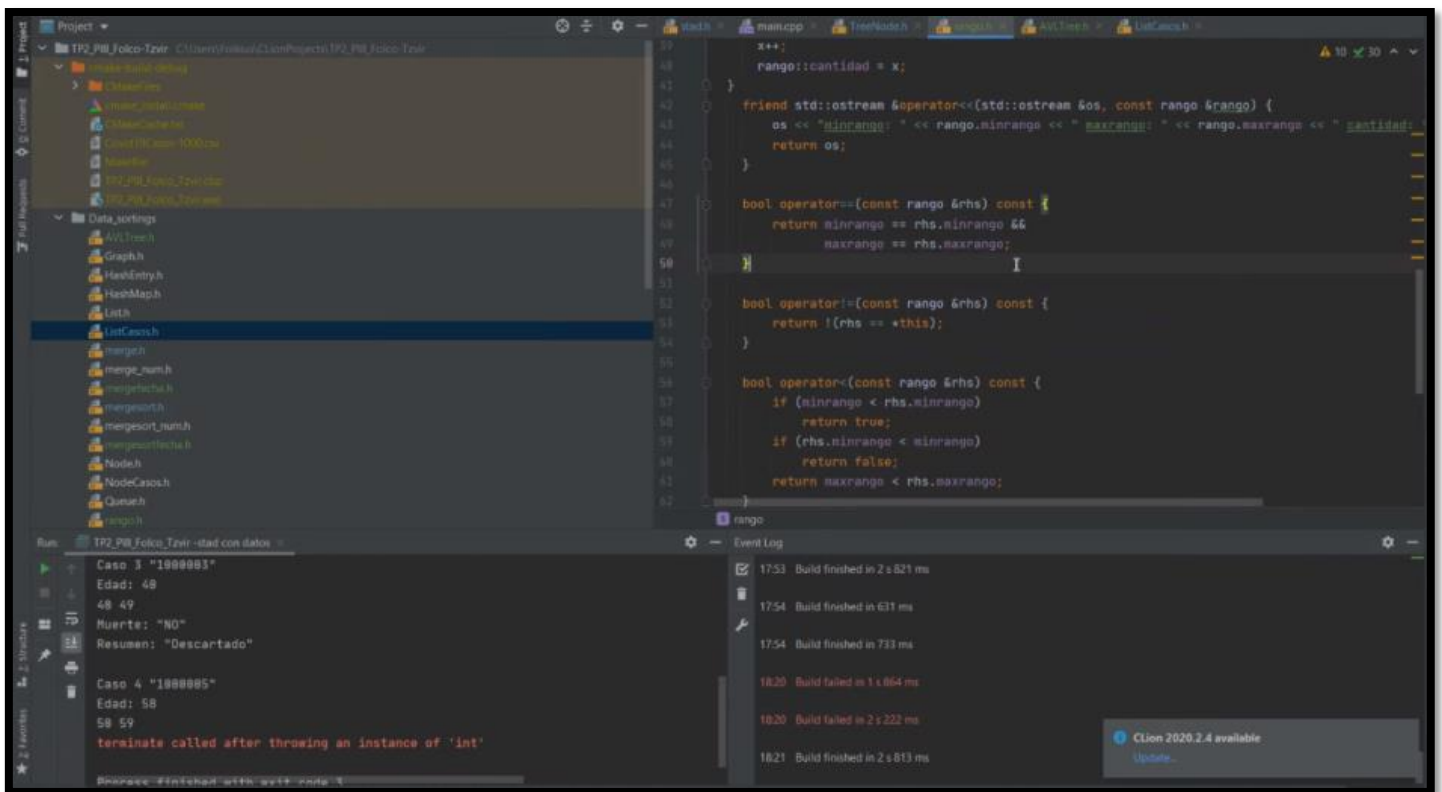
Sin embargo, el problema continúa, y se denota que la complicación es que el programa tiene dificultades a la hora de realizar el ordenamiento por las barras “\” presentes en el archivo .CSV. Para reparar el error, se agrega una nueva librería (“bits/stdc++”), que se ocupa de los espacios, permitiendo arreglar la fecha mayor o igual a la que manda el usuario (en imagen la línea donde se aplica el contenido de librería).

```
List<Paciente> sorteado = mergesortcui(delcui);

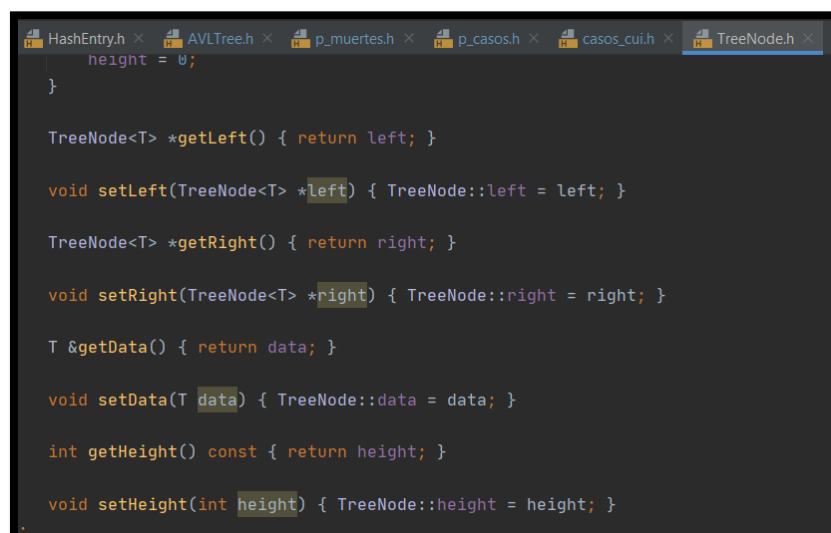
if (x == "1970-01-01") {
    sorteado.print();
    return 0;
}
string aux = "\"" + x + "\"";
sorteado.printf(aux);
return 0;
}
```

Para el caso de “stad.h”, se procede a borrar funciones innecesarias y aplicar los .h correspondientes para obtener el árbol binario balanceado. Se realiza una nueva clase “rango.h” para los casos estadísticos de “Cantidad de infectados por rango etario (rango de 10 años)” y “Cantidad de muertes por rango etario (rango de 10 años)”.

El inconveniente que se genera al momento de intentar aplicar el árbol binario balanceado es que se presenta una excepción cuando el dato presentado es igual al que ya se encuentra en el árbol binario balanceado. Como forma de afrontar el problema, se agarra la excepción y se intenta contabilizar la igualdad.

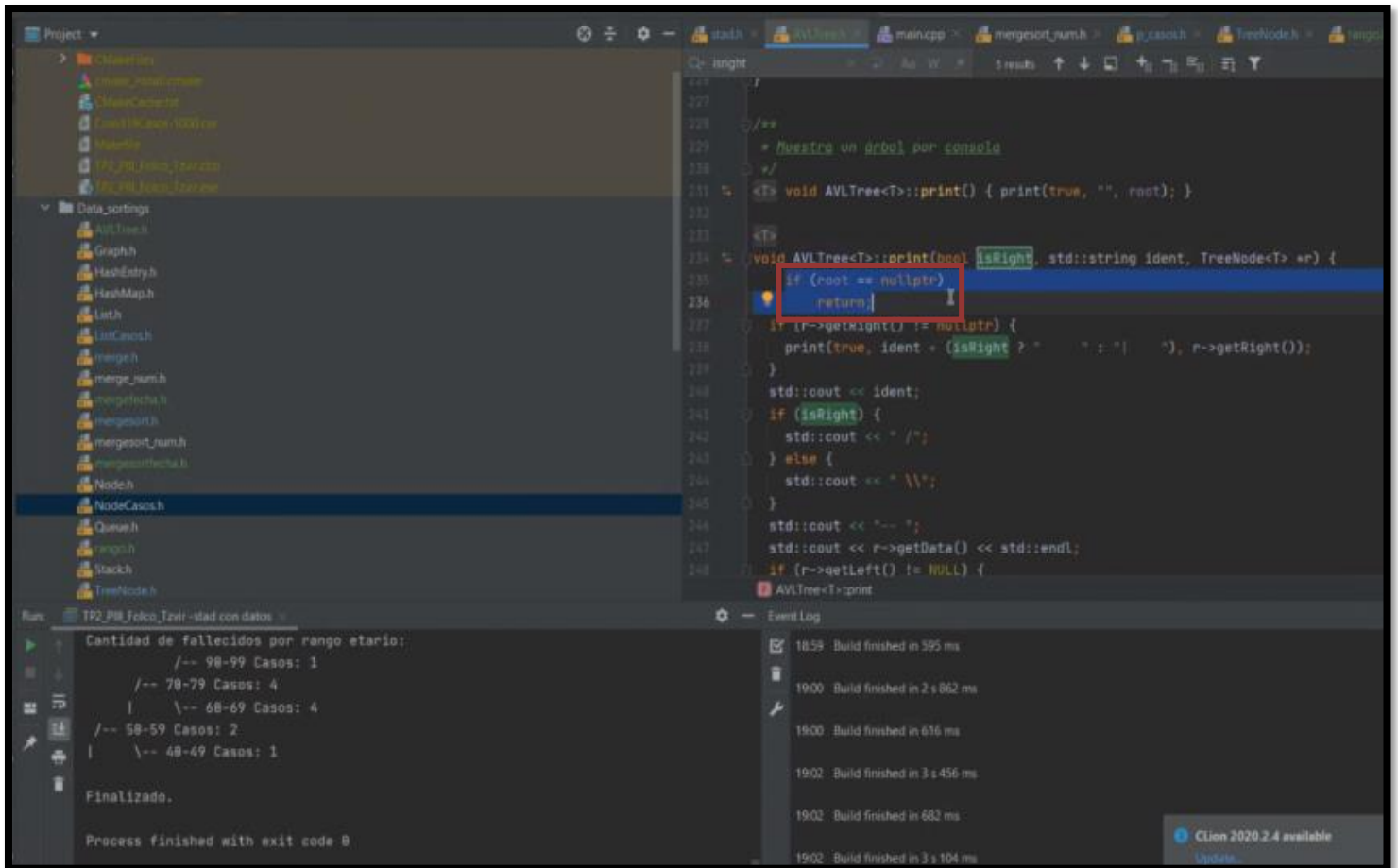


Sin embargo, no es hasta el agregado de & en **"TreeNode.h"** (para que se referencie al mismo dato y no una copia) que funciona el árbol binario balanceado.



Esto permite el armado del árbol binario balanceado.

También se presenta la complicación de que, al generarse dos árboles binarios balanceados distintos (para fallecidos e infectados) no permite leer ambos, pero al agregar una línea en el algoritmo que indique si el mismo se encuentra vacío o no logra solucionar el tema.



Finalmente, se realiza la documentación del programa, dando por finalizado la etapa de construcción y implementación del algoritmo.

Tarea: analizar refinamiento del programa y performance.

22/11: ÚLTIMOS DETALLES

Se realizan los últimos commits del programa, que se relacionan, optimización rápida (como armar funciones para no repetir código), desarrollo del README.md y correcciones.

CONCLUSIÓN

Como conclusión, puede expresarse que la realización de este trabajo práctico permitió la aplicación de conocimientos y temario visto en la cursada, logrando aplicarse formas de lectura y implementación de estructuras de datos variadas.

Para este trabajo, se tomó mucho en cuenta la forma y el comportamiento de las estructuras de datos, además de su longitud y performance, analizando tanto la ventaja como el costo de utilizar la misma.

El proceso para la construcción de los argumentos fue un proceso lento, ya que a veces se requería de días para el armado de los métodos. En momentos aparecían dificultades que se presentaban y que requerían desde un simple cambio de una línea en el comando al rearmado completo de una función. Sin embargo, a pesar de esos contratiempos, pudo llevarse a cabo la visualización y la muestra de datos con éxito.

Debido a complicaciones con el tiempo, detalles como utilización de tablas Hash o simplificación de código no pudieron ser posibles.

Finalmente, se puede decir, a modo personal, que este parcial fue una experiencia que a pesar de tener sus dificultades, fue enriquecedora. Este programa tiene elementos que pueden ser utilizados tanto para algún uso en la vida cotidiana, como para inspiración en la aplicación de otros códigos.