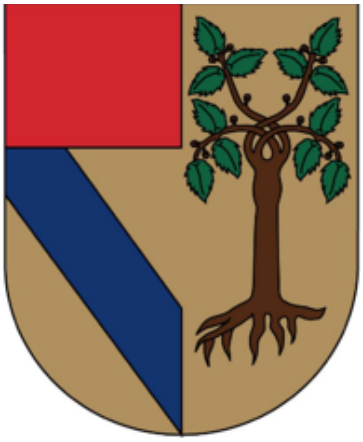


UNIVERISIDAD PANAMERICANA

CAMPUS MIXCOAC

Programación Avanzada

Informe

Segundo Parcial

Alumnos:

Santiago Uribe Alcántara

Enrique Ulises Báez Gómez Tagle

Guillermo Hernández Sosa

5 de abril de 2022

Introducción: El origen del proyecto se dio en clase. El profesor David Escobar Castillejos, nos pidió hacer un sistema de cafetería para nuestro segundo examen parcial. Nosotros como equipo, primero empezamos discutiendo cual iba a ser el método que usaríamos para poder trabajar todos como equipo y que camino íbamos a tomar para la creación de este código.

Nuestro método para trabajar en el código fue que cada clase, los tres, íbamos a estar avanzando en el código, usando una herramienta llamada GitHub, donde pudimos compartir el código y ver los cambios que la otra persona hizo.

Todos trabajamos correctamente, nos dividimos algunas partes del código para que nos diera tiempo de terminar el código, y muchas otras partes las hicimos todos juntos en clase. Funcionamos bien como equipo, muy bien coordinados, nos apoyamos en todo. Si alguien tenía alguna pregunta sobre como hizo una parte del código, la otra persona se tomaba el tiempo para explicarnos y que pudiéramos entender.

La conceptualización de cada clase fue la siguiente; Empezamos por definir dos tipos de clases, la primera que fue la clase de estudiante y la segunda, que fue la clase de administrador. Para estas clases lo que hicimos fue hacerlas públicas, para student definimos los siguientes aspectos:

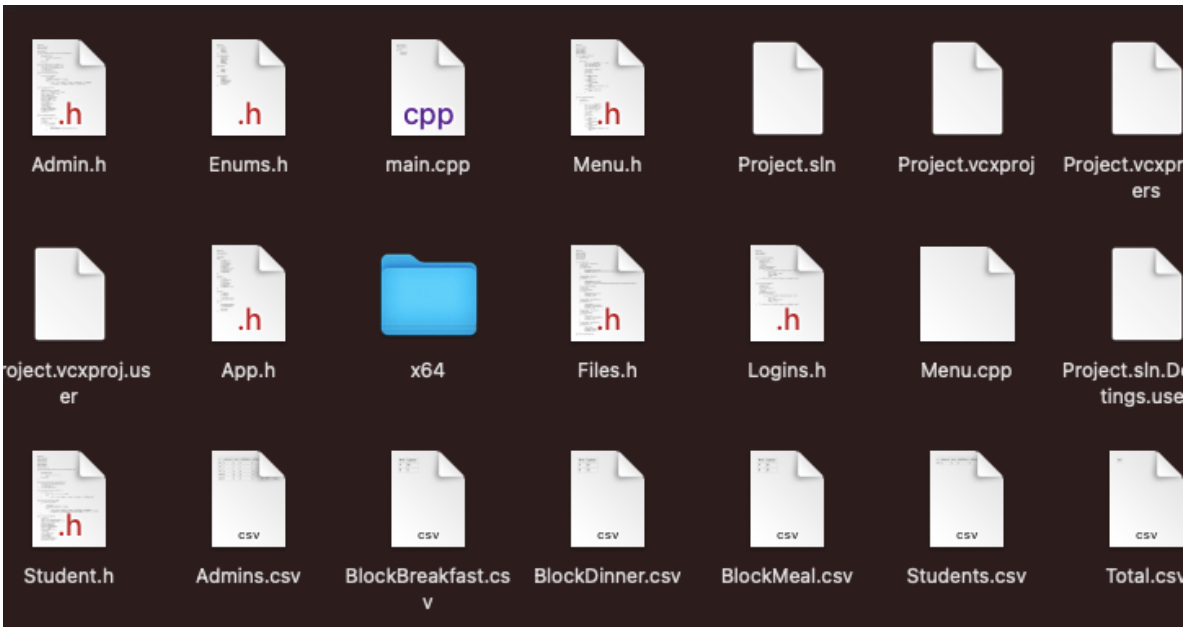
ID

1. Contraseña
2. Nombre
3. Segundo Nombre
4. Apellido
5. Edad
6. Género
7. Status
8. Desayuno
9. Comida
10. Cena

Para la clase de admin fueron las mismas opciones, pero sin los aspectos de desayuno, comida y cena.

Método:

Lo primero que empezamos a hacer fue pensar cuantos headers queríamos poner, y los dividimos en Enums, header de admin, header de students, el header de todos los menús que íbamos a necesitar, el header del sistema login, el header de files, el heder de app. Además de los headers, obviamente colocamos un main cpp donde mandamos a llamar todas las funciones para que pudiera correr el programa.



En el Header de Enums decidimos que íbamos a colocar 4 enumeradores, los cuales son: EnumStudent con sus respectivas variables que indican los atributos que estas tienen. Por ejemplo:

1. LogStud
2. AddStud
3. PrAStud
4. ExitStud

El segundo enumerador que pusimos fue el de EnumLoggedStudent donde las variables que decidimos colocar fueron las de registrarse en los 3 bloques de comida y las de dar de alta y dar de baja.

El tercer enumerador fue el de el Administrador, en el cual pusimos las indicaciones de registrar un administrador, agregar un administrador y el de imprimir los datos del administrador.

Por último, el enumerador de las opciones que tendría el administrador, una vez ya ingresado en el sistema. Las opciones que este tendría serían:

1. Modificar los bloques de comida
2. Modificar la capacidad en los bloques
3. Ver cada bloque
4. Poder dar de baja administradores y dar de alta también
5. Modificar la capacidad específica de cada comida.
6. Modificar los estudiantes en cada bloque
7. Los administradores ya registrados.

```
#pragma once

enum StudentMenu
{
    LogStud = 1,
    AddStud,
    PrAStud,
    ExitStudent
};

enum LoggedStudentMenu
{
    DisStudent = 1,
    EnaStudent,
    RegBreakfast,
    RegMeal,
    RegDinner,
    ExitStudentLog
};

enum AdminMenu
{
    LogAdm = 1,
    AddAdm,
    PrAAdm,
    ExitAdmin
};

enum LoggedAdminMenu
{
    DisAdmin = 1,
    EnaAdmin,
    EdTotCap,
    EdSpecMealBlockCap,
    HowManyStudsBlock,
    WhichStudsBlock,
    ExitAdminLog
};
```

Nuestro segundo header fue el del administrador. Ahí pusimos todo lo que debería de llevar un administrador.

Empezamos por definir una clase Admin con los valores públicos, estos valores son:

1. Un ID
2. Contraseña
3. Nombre
4. Primer Apellido
5. Segundo Apellido
6. Edad
7. Género

Después de esto, hicimos un barrido con CheckAdminID para ver si el ID era correcto y que no se hubiera repetido. En los siguientes bloques de código lo que hicimos fue habilitar un administrador y deshabilitar un administrador. En estos les pedimos al inicio que ingrese su ID, después, si el ID es correcto, se despliega el mensaje de: “Administrador habilitado correctamente”, de lo contrario se desplegará un mensaje “El ID del administrador no se encuentra”. Con el bloque de código de deshabilitar admin es prácticamente lo mismo que en el de arriba.

```
inline bool check_admin_id(const int id, const vector<admin> list)
{
    bool used = false;
    for (auto& i : list)
    {
        if (i.id == id) used = true;
        break;
    }
    return used;
}

inline int get_admin_id(const vector<admin> list)
{
    const int n = rand() % 100;
    const bool used = check_admin_id(n, list);
    if (!used) return n;
    return get_admin_id(list);
}
```

El siguiente bloque es nada más desplegar los datos que el admin ya tuvo que haber ingresado. Esto lo hicimos solo con un ciclo *for* y puro *cout*.

```
inline void print_admins(const app App)
{
    for (auto& i : App.admins)
    {
        string status;
        if (i.status == "true") status = "Active";
        else status = "Inactive";
        cout << "----------------------" << endl;
        cout
            << "ID: " << i.id << "\tName: " << i.name << "\tMiddle Name: " << i.middleName
            << "\tLast Name: " << i.lastName << "\tStatus: " << status << endl;
    }
}
```

Y para la parte final del header de admin, pusimos los de AddAdmin y la función de Log Admin.

En AddAdmin, creamos un variable temporal de Admin, y después empezamos a desplegar los mensajes al usuario (el que sería el administrador), para que empiece a llenar los datos que le estamos pidiendo, como:

1. Su ID
2. Nombre
3. Género
4. Edad
5. Apellidos
6. Contraseña

```
inline void add_admin(app& App)
{
    admin temp;
    temp.id = get_admin_id(App.admins);
    cout << " ------ Add Admin ------ " << endl;
    cout << "ID: " << temp.id << endl;
    cout << "Name: ";
    getline(cin, temp.name);
    cout << "Middle Name: ";
    getline(cin, temp.middleName);
    cout << "Last Name: ";
    getline(cin, temp.lastName);
    cout << "Age: ";
    cin >> temp.age;
    cin.ignore();
    cout << "Gender (M-F): ";
    cin >> temp.gender;
    cin.ignore();
    cout << "PASSWORD: ";
    getline(cin, temp.password);
    App.admins.push_back(temp);

    system("clear");
}
```

En el Log Admin, empezamos con un booleano llamado *run*, con valor verdadero. Usamos un ciclo while (run) y después enseñamos el menú que solo se le desplegará al que sea administrador. El menú tiene las siguientes opciones:

1. Deshabilitar Admin
2. Habilitar Admin
3. Editar capacidad total
4. Editar un bloque de comida específico
5. Ver cuántos estudiantes hay en cada bloque
6. Ver que estudiantes están en los bloques

Después el administrador, mete la opción deseada y usamos un ciclo switch con la opción de default para que despliegue “Opción invalida” en caso de que el administrador haya escrito alguna opción que no estaba en el menú.

También pusimos las opciones de enable admin y las de disable admin. Par esto solo preguntamos el ID, hacemos un barrido para que busque los IDs y pues dependiendo del bloque en el que esté, va a salir si lo habilitamos o no.

Después pusimos las funciones de editar la capacidad total y específicamente en cada bloque de comida.

Esto se logró con, primero preguntándole al admin que ingresara un numero y después ese numero se iba a guardar en la base de datos y ya después desplegamos unos case para cada bloque de comida

```
inline void edit_total_capacity(app& App)
{
    cout << "-------- EDIT TOTAL CAPACITY --------" << endl;
    cout << "Enter new total capacity: ";
    cin >> App.total_students;
    cin.ignore();
    cout << "Total Capacity successfully updated" << endl;
}

inline void edit_specific_meal_capacity(app& App)
{
    int option;
    cout << "-------- EDIT SPECIFIC MEAL CAPACITY --------" << endl;

    cout << "1) Breakfast" << endl;
    cout << "2) Meal" << endl;
    cout << "3) Dinner" << endl;

    cout << "Select your option: ";
    cin >> option;
    cin.ignore();

    switch (option)
    {
    case 1:
    {
        cout << "Enter new breakfast capacity: ";
        cin >> App.breakfast.total;
        cin.ignore();
        cout << "Breakfast capacity successfully updated" << endl;
        break;
    }
    case 2:
    {
        cout << "Enter new meal capacity: ";
        cin >> App._meal.total;
        cin.ignore();
        cout << "Meal capacity successfully updated" << endl;
        break;
    }
```

También hicimos la función de cuantos estudiantes por bloque. Aquí al inicio fue con puro cout y le preguntamos que por favor ingrese una opción de comida y en que bloque, y ya

con los cases fuimos dando forma para que le desplegara según lo que el estudiante ingresó, cuantos estudiantes por bloque había.

```
inline void how_may_students_block(app& App)
{
    int hour;
    string block;
    int capacity = 0;
    cout << "-------- HOW MANY STUDENTS IN BLOCK--------" << endl;
    cout << "1) Breakfast" << endl;
    cout << "2) Meal" << endl;
    cout << "3) Dinner" << endl;

    cout << "Select your option: ";
    cin >> hour;
    cin.ignore();

    cout << "Block A" << endl;
    cout << "Block B" << endl;
    cout << "Select your option: ";

    cin >> block;
    cin.ignore();

    switch (hour)
    {
    case 1:
    {
        if (block == "A")
        {
            capacity = App.breakfast.total_a;
            cout << "Breakfast capacity successfully updated" << endl;
        }
        else if (block == "B")
        {
            capacity = App.breakfast.total_b;
            cout << "Breakfast capacity successfully updated" << endl;
        }
        else
        {
            cout << "Invalid option" << endl;
```

Nosotros creamos un header llamado app.

En este header están incluidas todas las clases y estructuras. Además de que creamos una estructura llamada app, en esta estructura llamada app metimos todo lo que ya había mencionado, para que estén todos los datos, pero dentro de una estructura nada más.

```
class student
{
public:
    int id{};
    string password{};

    string name;
    string middleName;
    string lastName;
    int age{};
    string gender{};
    string status = "true";

    string breakfast{};
    string _meal{};
    string dinner{};
};

class admin
{
public:
    int id{};
    string password{};

    string name;
    string middleName;
    string lastName;
    int age{};
    string gender{};
    string status = "true";
};
```

```
struct app
{
    vector<student> students;
    vector<admin> admins;
    int total_students{};

    meal breakfast;
    meal _meal;
    meal dinner;
};
```

Para Header de Student, empezamos con puros inline para ver que el ID del estudiantes esté bien y no haya problemas después y también para imprimir los datos del estudiantes, como:

1. Nombre
2. ID
3. Apellido

```cpp
using namespace std;

inline bool check_student_id(const int id, const vector<student>& list)
{
    bool used = false;
    for (auto& i : list)
    {
        if (i.id == id) used = true;
        break;
    }
    return used;
}

inline int get_student_id(const vector<student>& list)
{
    const int n = rand() % 100;
    const bool used = check_student_id(n, list);
    if (!used) return n;
    return get_student_id(list);
}

inline void print_students(vector<student>& list)
{
    for (auto& i : list)
    {
        cout << "----------------------" << endl;
        cout
            << "ID: " << i.id << "\tName: " << i.name << "\tLast Name: " << i.lastName << end
    }
}
```

Le sigue una opción booleana que verifica que el ID esté correcto y que no esté repetido.

El siguiente bloque que hicimos fue la función de AddStudent. Ahí también creamos una variable temporal llamada Student, después ya le desplegamos al estudiante los mensajes con los datos que queremos que llene, incluyendo su contraseña.

```
inline void add_student(app& App)
{
    student temp;
    temp.id = get_student_id(App.students);
    cout << " ------ Add Student ------ " << endl;
    cout << "ID: " << temp.id << endl;
    cout << "Name: ";
    getline(cin, temp.name);
    cout << "Middle Name: ";
    getline(cin, temp.middleName);
    cout << "Last Name: ";
    getline(cin, temp.lastName);
    cout << "Age: ";
    cin >> temp.age;
    cin.ignore();
    cout << "Gender (M-F): ";
    cin >> temp.gender;
    cin.ignore();
    cout << "PASSWORD: ";
    getline(cin, temp.password);
    App.students.push_back(temp);

    system("clear");
}
```

Creamos una función para poder imprimir los datos que el estudiante ingresó y poder verlos en la terminal.

```
inline void print_students(const app App)
{
    for (auto& i : App.students)
    {
        string status;
        if (i.status == "true") status = "Active";
        else status = "Inactive";
        cout << "----------------------" << endl;
        cout
            << "ID: " << i.id << "\tName: " << i.name << "\tMiddle Name: " << i.middleName
            << "\tLast Name: " << i.lastName << "\tStatus: " << status << "\tBreakfast: " << i.breakfast
            << "\tMeal: " << i._meal << "\tDinner: " << i.dinner << endl;
    }
}
```

Para la parte de habilitar estudiantes y deshabilitar estudiantes, son casi lo mismo que las de administrador. Empezamos creando un *bool found = false* y preguntamos por el ID del estudiante que queremos habilitar, después usamos un ciclo *for* para poder encontrar el ID que nos pidieron, si no se encontró el ID, entonces se va a desplegar un mensaje de fracaso en el intento. Y para deshabilitar es lo mismo, solo que cambiando el status a falso.

```
inline void enable_student(app& App)
{
    bool found = false;
    cout << "ID of Student to enable: ";
    int is;
    cin >> is;
    cin.ignore();
    for (auto& i : App.students)
    {
        if (is == i.id)
        {
            found = true;
            i.status = "true";
            cout << "Student status successfully updated" << endl;
        }
    }
    if (!found) cout << "Student ID not found" << endl;
}

inline void disable_student(app& App)
{
    bool found = false;
    cout << "ID of Student to disable: ";
    int is;
    cin >> is;
    cin.ignore();
    for (auto& i : App.students)
    {
        if (is == i.id)
        {
            found = true;
            i.status = "false";
            cout << "Student status successfully updated" << endl;
        }
    }
    if (!found)
    {
        cout << "Student ID not found" << endl;
    }
}
```

También pusimos tres funciones más las cuales son:

1. Register breakfast
2. Register meal
3. Register Dinner

Para estas partes, lo que hicimos fue empezar con un if, dentro del if mandamos a llamar a la app.dinner.total y si era mayor o igual a la app.total.students pues se despliega el mensaje de que ya no había espacio, de lo contrario, le pedimos su ID, que eligiera el bloque y creamos unas líneas para que la capacidad fuera aumentando.

```
inline void register_dinner(app& App)
{
    bool found = false;
    string block;

    if (App.dinner.total >= App.total_students)
    {
        cout << "No more students can register for dinner" << endl;
        return;
    }
    cout << "ID of Student: ";
    int is;
    cin >> is;
    cin.ignore();
    for (auto& student : App.students)
    {
        if (is == student.id)
        {
            App.dinner.total++;
            found = true;
            cout << "Select block: ";
            cin >> block;
            if (block == "A" || block == "B")
            {
                student.dinner = block;
                if (student.dinner == "A")
                {
                    App.dinner.total_a++;
                    App.dinner.list_a.push_back(student);
                    break;
                }
                App.dinner.total_b++;
                App.dinner.list_b.push_back(student);
                break;
            }
        }
        cout << "Invalid block" << endl;
        break;
```

La última parte del header de estudiantes es un LogStudent. Aquí igual empezamos con *bool run=true* y con ciclo *while(run)*, después, dentro del ciclo while, se despliega un menú solo para estudiantes, donde tenemos las opciones de:

1. Deshabilitar Estudiante
2. Habilitar Estudiante
3. Registra Desayuno
4. Registrar Comida
5. Registrar Cena

Después de que la computadora lea la opción ingresada usamos un switch con la opción ingresada con los distintos casos ya especificados en el menú, y cada caso hace lo que está indicado en él.

```
inline void logged_student(app& App)
{
    bool run = true;
    int option;

    while (run)
    {
        cout << "-------- LOGGED STUDENT MENU -------- " << endl;
        cout << "1) Disable student" << endl;
        cout << "2) Enable student" << endl;
        cout << "3) Register Breakfast" << endl;
        cout << "4) Register Meal" << endl;
        cout << "5) Register Dinner" << endl;
        cout << "6) Exit" << endl;

        cout << "Select your option: ";
        cin >> option;
        cin.ignore();

        switch (option)
        {
        case DisStudent:
        {
            cout << "Disabling student" << endl;
            disable_student(App);
            break;
        }

        case EnaStudent:
        {
            cout << "Enabling student" << endl;
            enable_student(App);
            break;
        }
```

El siguiente header es el de files.h

Empezamos con una función Inline que quiere decir que cuando compilamos, no las llamamos en el código objeto o sea no llama a la función, sino se crea una copia de ella y se inserta en el código. Para el primer Inline hicimos la función de *make files*. Aquí nada mas pusimos que parte o que funciones queríamos convertir a archivos csv y que es lo que queríamos que estos archivos tuvieran y eso fue todo.

```
inline void make_files()
{
    string filename = "Students.csv";
    ifstream ifile;
    ifile.open(filename);
    if (!ifile)
    {
        ifile.close();
        ofstream myFileOutput(filename);
        myFileOutput <<
            "ID,Password,Name,Middle_Name,Last_Name,Age,Gender,Status,Breakfast,Meal,Dinner\n";
        myFileOutput.close();
    }

    string filename2 = "Admins.csv";
    ifstream ifile2;
    ifile2.open(filename2);
    if (!ifile2)
    {
        ifile2.close();
        ofstream myFileOutput(filename2);
        myFileOutput <<
            "ID,Password,Name,Middle_Name,Last_Name,Age,Gender,Status,Breakfast,Meal,Dinner\n";
        myFileOutput.close();
    }
```

```
inline void read_admins(app& App)
{
    string line, colname;
    vector<string> headers;
    string value;

    ifstream myFileInput("Admins.csv");
    if (!myFileInput.is_open()) throw runtime_error("Could not open file");
    if (myFileInput.good())
    {
        // Extract the first line in the file
        getline(myFileInput, line);

        // Create a stringstream from line
        stringstream ss(line);

        // Extract each column name
        while (getline(ss, colname, ','))
        {
            headers.push_back(colname);
        }
    }

    while (getline(myFileInput, line))
    {
        // Create a stringstream of the current line
        stringstream ss(line);

        int colIdx = 0;
        int _id, _age;
        string _password, _name, _middle_name, _last_name, _gender, _status;

        while (getline(ss, value, ','))
        {
            switch (colIdx)
```

Y lo mismo hicimos para cada bloque de comida, un ejemplo es la cena:

```
inline void read_blocks_dinner(app& App)
{
    string line, colname;
    vector<string> headers;
    string value;
    ifstream myFileInput("BlockDinner.csv");
    if (!myFileInput.is_open()) throw runtime_error("Could not open file");
    if (myFileInput.good())
    {
        // Extract the first line in the file
        getline(myFileInput, line);

        // Create a stringstream from line
        stringstream ss(line);

        // Extract each column name
        while (getline(ss, colname, ','))
        {
            headers.push_back(colname);
        }
    }
    while (getline(myFileInput, line))
    {
        // Create a stringstream of the current line
        stringstream ss(line);

        int colIdx = 0;
        string _block;
        int _capacity;

        while (getline(ss, value, ','))
        {
            switch (colIdx)
            {
            case 0:
                block = value;
```

También creamos un a función inline para actualizar los bloques de comida y que se vieran reflejados en al archivo extremo

```
inline void update_block(const app App)
{
    ofstream myFileOutput("BlockBreakfast.csv");
    ofstream myFileOutput2("BlockMeal.csv");
    ofstream myFileOutput3("BlockDinner.csv");
    myFileOutput << "Block,Capacity\n";
    myFileOutput << "A" << "," << App.breakfast.total_a << "\n"
        << "B" << "," << App.breakfast.total_b;
    myFileOutput2 << "Block,Capacity\n";
    myFileOutput2 << "A" << "," << App._meal.total_a << "\n"
        << "B" << "," << App._meal.total_b;
    myFileOutput3 << "Block,Capacity\n";
    myFileOutput3 << "A" << "," << App.dinner.total_a << "\n"
        << "B" << "," << App.dinner.total_b;
}
```

El siguiente header que hicimos fue el de Logins

El primero que hicimos fue el de Log admin. Aquí le pedimos a al admin que por favor ingresará un ID y después que ingresará una contraseña. La máquina lee estos datos y los guarda.

Después con un ciclo *for* y con *if* indicamos que si la contraseña y el ID coinciden con los que estaban registrados, se cambiaría la bandera a true, junto con el mensaje de Welcome. Sino pues no lo dejaría entrar.

Y para el Login de students fue exactamente el mismo procedimiento

```cpp
#pragma once

#include "App.h"
#include "Admin.h"
#include "Student.h"


inline void login_admin(app& App)
{
    bool found = false;
    int id;
    string password;
    cout << "Enter your ID: ";
    cin >> id;
    cin.ignore();
    cout << "Enter your password: ";
    getline(cin, password);
    for (auto& i : App.admins)
    {
        if (i.id == id && i.password == password && i.status == "true")
        {
            found = true;
            cout << "Welcome " << endl;
            logged_admin(App);
            break;
        }
    }
    if (!found) { cout << "Invalid ID or password or user disabled" << endl; }
}
```

Nuestro último header fue el de los menús.

En este básicamente lo que hicimos fue colocar nuestros tres menús, el de estudiantes, el de administradores y el menú principal.

Para los menús, la lógica que seguimos fue muy sencilla, pues solo consiste en agregar una variable llamada opción y un booleano con valor true para que corra ese bloque de código.

Después adentro del while, ya pusimos las opciones que tendría el menú, esto dependiendo de que menú fuera y después con un switch pusimos los casos con las opciones del menú.

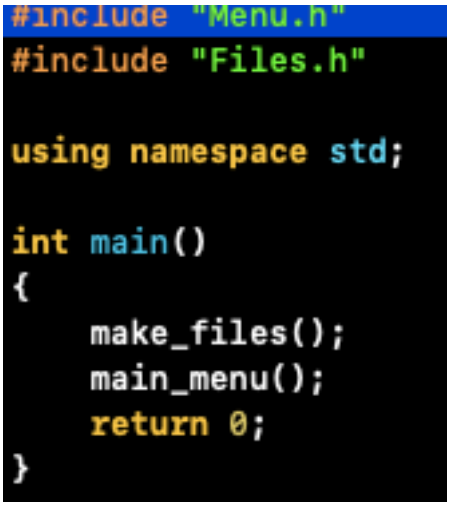
```cpp
inline void admin_menu(app& App)
{
    int option;
    bool run = true;

    while (run)
    {
        cout << "-------- ADMIN MENU --------" << endl;
        cout << "1) Login" << endl;
        cout << "2) Add Admin" << endl;
        cout << "3) Print All" << endl;
        cout << "4) Exit" << endl;

        cout << "Select your option: ";
        cin >> option;
        cin.ignore();

        switch (option)
        {
        case LogAdm:
            login_admin(App);
            break;
        case AddAdm:
            add_admin(App);
            break;
        case PrAAdm:
            print_admins(App);
            break;
        case ExitAdmin:
            cout << "Return to Main Menu" << endl;
            run = false;
            break;
        default:
            cout << "Invalid option";
            break;
        }
    }
}
```

Para terminar nuestro programa, tenemos un main cpp, en donde colocamos dentro del main las funciones de make files y el menú principal.



```
#include "Menu.h"
#include "Files.h"

using namespace std;

int main()
{
    make_files();
    main_menu();
    return 0;
}
```

Resultados: Si, la simulación se está ejecutando correctamente. Lo primero que hace es pedir la información básica, como, por ejemplo: id, nombre, etc. Después lo que hace el código es que despliega dos menús. Uno para estudiantes y otro para directivos/administradores. Los que administran tiene opciones diferentes a los estudiantes, como lo son, decidir sobre los menús, cambiar el cupo máximo del turno, etc. Después, a los estudiantes se les desplegará el menú con todas las opciones que hay para comer y para que bloque del día lo desea, ya sea de mañana, cena o comida.

Durante el proceso de creación del código, sufrimos unos atrasos ya que los aparatos no funcionaban bien unos días, en otra ocasión teníamos problemas con los programas que utilizamos como GitHub, Xcode o Visual Studio Code. Esos problemas pequeños nos atrasaron un poco, pero al final los pudimos resolver y el código si corre bien.

Discusión:

Guillermo: Este proyecto me gustó mucho. Fue un gran desafío para mí. Nunca había hecho un código que estuviera basado en la vida real y sobre todo un este proyecto que fue el sistema de una cafetería, donde una cafetería ocupa muchas opciones y muchas cosas para que pueda funcionar bien. Yo creo que nuestro equipo hizo un gran trabajo y pudimos hacer y correr un buen programa.

Enrique: Fue un gran reto, cuando hicimos la planeación del código, sin problemas diseñamos un programa idóneo, pero los errores con los entornos de desarrollo y el control de versiones nos atrasaron muchísimo. La aplicación es interesante, ya que se adapta a un caso de la vida cotidiana que nosotros como estudiantes mismo vivimos.

Santiago: Este proyecto fue una prueba muy grande de los aspectos en los que tengo que practicar más y mejorar. El código quedó muy padre; se siente muy bien cuando un código funciona de la forma en la que debe.

Discusión Global

Este fue un gran reto para nosotros. Nos gustó mucho este proyecto, pues representaba un problema, un código que en la vida real sí se necesita, sí sirve. Este proyecto nos puso a prueba, porque si bien teníamos una idea de como poder ejecutar y crearlo, hubo muchos errores y fallos y muchas veces nos estresábamos y no podíamos seguir con el código.

Nos encontramos con bastantes errores durante el desarrollo del código, sin embargo, os pudimos resolver. Los resolvimos investigando en internet, nos juntábamos los tres integrantes para discutir como y hacer una lluvia de ideas para pode resolver los problemas que hacían los bugs.

Para terminar. Podemos acordar los integrantes de este grupo, que, en efecto, fue un gran desafío para todos nosotros. Nos estresamos, pero también nos aliviábamos cada vez que los lográbamos resolver. Nos gustó el código, fue retador y otro punto por el que nos gustó el código fue por que ya es un código que se asemeja más a la realidad, a la vida laboral.