“Actividad: Laboratorios del Módulo 1 de C++ Essentials 2”

Materia: Programación orientada a objetos

Profesor: Manuel Balderas Victoria

Carrera: Ingeniería en software y minería de datos

Semestre: 3°ro

Alejandro Guzmán Cabrales (5110517)

15/09/25

Laboratorio “1.0.7 Classes and Objects in C++”

```cpp
#include <iostream>
#include <string>

using namespace std;

class Person
{
public:
    string name;
    int    age;
    // Your code here
    double height;
    double weight;
};

void print(Person* person)
{
    cout << person->name << " is " << person->age
        << " years old, they're " << person->height
        << " meters tall and weigh " << person->weight << " Kilograms" << endl;
}
```

```cpp
int main()
{
    Person person;
    person.name = "Harry";
    person.age = 23;
    person.height = 1.76;
    person.weight = 78.6;

    cout << "Meet " << person.name << endl;
    print(&person);

    // Your code here

    return 0;
}
```

En este laboratorio empezamos a familiarizarnos con las clases y objetos, el objetivo siendo añadir algunos atributos e imprimirlos en pantalla. Este código en C++ define una clase llamada **Person**, la cual representa a una persona con atributos básicos: **nombre** (string name), **edad** (int age), **estatura** (double height) y **peso** (double weight). En el main, se crea un objeto person con valores asignados a cada atributo para después imprimirlos en la consola mediante la función **print**, que recibe un puntero a Person.

Output:

```
Meet Harry
Harry is 23 years old, they're 1.76 meters tall and weigh 78.6 Kilograms

C:\Users\aleja\source\3er Parcial\Laboratorio POO c++\x64\Debug\Laborato
 (0x0).
Press any key to close this window . . .
```

Laboratorio “1.0.8 Restricting access to object data”

```cpp
#include <iostream>
#include <string>

using namespace std;

class Square
{
public:
    Square(double side);

    void set_side(double side)
    {
        this->side = side;
        this->area = side * side;
    }

    void print()
    {
        cout << "Square: side = " << this->side << " area = " << this->area << endl;
    }

    // Your code here
private:
    double    side;
    double    area;
};

Square::Square(double side)
{
    set_side(side);
}
```

```cpp
int main()
{
    Square s(4);


    s.print();

    s.set_side(2.0);
    s.print();

    s.set_side(-33.0);
    s.print();

    return 0;
}
```

En este laboratorio utilizamos la keyword **private** para restringir el acceso a los atributos de una clase **Square**. La clase square tiene de atributo Int **Side** e un Int **Area**. Para acceder a él, tenemos el método **set_side**(double side) que cambia el valor del atributo y recalcula el valor de “área” del objeto. En la función int **main**() creamos un objeto **Square** de lado 4, lo imprimimos y cambiamos su atributo dos veces para revisar resultados.

Output:

```
Square: side = 4 area = 16
Square: side = 2 area = 4
Square: side = -33 area = 1089

C:\Users\aleja\source\3er Parcial\Laborato
 (0x0).
Press any key to close this window . . .
```

Laboratorio “1.0.9 Obtaining derived data from an object”

Clase AdHocSquare:

```cpp
#include <iostream>

class AdHocSquare
{
public:
    AdHocSquare(double side);
    void set_side(double side);
    double get_area() { return side * side; }
private:
    double    side;
};

AdHocSquare::AdHocSquare(double side)
{
    this->side = side;
}

void AdHocSquare::set_side(double new_side)
{
    if (new_side > 0)
    {
        side = new_side;
    }
}
```

Clase LazySquare:

```cpp
class LazySquare
{
public:
    LazySquare(double side);
    void set_side(double side);
    double get_area();

private:
    double    side;
    double    area;
    bool      side_changed;
};

LazySquare::LazySquare(double side)
{
    this->side = side;
    area = side * side;
    side_changed = false;
}

void LazySquare::set_side(double new_side)
{
    if (side < 0)
    {
        return;
    }
    if (side != new_side) {
        side = new_side;
        side_changed = true;
    }
}
double LazySquare:: get_area()
{
    if (side_changed)
    {
        area = side * side;
    }

    return area;
};
```

Función main():

```cpp
int main()
{
    AdHocSquare s1(4);
    std::cout << "Square area = " << s1.get_area() << std::endl;

    s1.set_side(6);
    std::cout << "Square area = " << s1.get_area() << std::endl;


    LazySquare s2(6);
    std::cout << "Square area = " << s2.get_area() << std::endl;

    s2.set_side(4);
    std::cout << "Square area = " << s2.get_area() << std::endl;

    return 0;
}
```

Output:

```
Square area = 16
Square area = 36
Square area = 36
Square area = 16

C:\Users\aleja\source\3er Parcial\Laborato
 (0x0).
Press any key to close this window . . .
```

En este código utilizamos dos formas de **obtener datos derivados** (el área)

En **AdHocSquare** se recalcula directamente cada vez que se solicita a partir del lado, mientras que en **LazySquare** el área se guarda y solo se actualiza cuando el lado cambia mediante un **buleano** que se actualiza cada vez que el lado cambie de valor, evitando cálculos innecesarios al acceder repetidamente.

Laboratorio “1.0.10 Classes and objects: ShopItemOrder”

Clase ShopItemOrder y Función main():

```cpp
#include <iostream>

class ShopItemOrder
{
    // Write your code here
private:
std::string item_name;
double item_price;
int items_ordered;

public:
    //Constructor
    ShopItemOrder(std::string item_name, double item_price, int items_ordered);

    //Getters y setters
    void set_item_name(std::string item_name) { this->item_name = item_name; }
    std::string get_item_name() { return item_name; }

    void set_item_price(double item_price) { this->item_price = item_price; }
    double get_item_price() { return item_price; }

    //Métodos
    double get_total_order() {return item_price * items_ordered;}

    void print_order();
};

ShopItemOrder::ShopItemOrder(std::string item_name, double item_price, int items_ordered)
{
    this->item_name = item_name;
    this->item_price = item_price;
    this->items_ordered = items_ordered;
}

void ShopItemOrder::print_order()
{
    std::cout << items_ordered << " " << item_name << " // $" << get_total_order() << std::endl;
}

int main()
{
    ShopItemOrder Pozol("Pozol bien frío", 35, 2);
    ShopItemOrder Hotcakes("Pancakes", 30, 5);

    Pozol.print_order();
    Hotcakes.print_order();

}
```

Output:

```
2 Pozol bien frÝo // $70
5 Pancakes // $150

C:\Users\aleja\source\3er Parcial\Laborat
0 (0x0).
Press any key to close this window . . .
```

En este laboratorio tenemos la instrucción de crear nuestra propia clase que cumpla con lo siguiente:

- Sea llamada **ShopItemOrder**.
- Almacene la sig información:
    - item name;
    - item unit price;
    - number of items ordered.
- Y tenga los siguientes métodos de acceso:
    - getters y setters para todos los atributos.
    - devolver el total de la orden.
    - Imprimir la orden.

Lo cual aplicamos a la medida en el código, cumpliendo con los atributos y los métodos.

Laboratorio “1.2.12 – 1.2.14 Flight booking system”

Clase FlightBooking:

```cpp
#include <iostream>

class FlightBooking {
private:
    int id;
    int capacity;
    int reserved;

public:
    FlightBooking(int id, int capacity, int reserved);
    FlightBooking();
    int getID();
    void printStatus();
    bool reserveSeats(int number_of_seats);
    bool cancelReservations(int number_of_seats);
};

FlightBooking::FlightBooking(int id, int capacity, int reserved)
{
    // Save data to members
    this->id = id;
    this->capacity = capacity;
    if (reserved <= capacity + capacity * .05)
    {
        this->reserved = reserved;
    }
    else
    {
        std::cout << "The reserved seats can not be higher than 105% the plane capacity"<<std::endl;
        this->reserved = 0;
    }
}
```

```cpp
FlightBooking::FlightBooking()
{
    // Initialize atributes
    this->id = 0;
    this->capacity = 0;
    this->reserved = 0;
}
int FlightBooking::getID()
{
    return this->id;
}

void FlightBooking::printStatus()
{
    // print report here
    double percent;
    if (capacity != 0) {percent = static_cast<double>((reserved) / capacity) * 100; }
    else { percent = 0; }
    if (id == 0) { return; }
    std::cout << "Flight [" << id << "] : " << reserved << "/" << capacity << " (" << percent << " %) seats taken" << std::endl;

}
bool FlightBooking::reserveSeats(int number_of_seats)
{
    // try to add reservations and return 'true' on success
    // keep the limits in mind
    if (number_of_seats + reserved > capacity * 1.05)
    {
        std::cout << "The reserved seats can not be higher than 105% the plane capacity" << std::endl;
        return false;
    }

    reserved += number_of_seats;
    return true;
}

bool FlightBooking::cancelReservations(int number_of_seats)
{
    // try to cancel reservations and return 'true' on success
    // keep the limits in mind

    if (number_of_seats > reserved)
    {
        std::cout << "The canceled seats can not be higher than the reserved seats." << std::endl;
        return false;
    }

    reserved -= number_of_seats;
    return true;
}
```

Función main():

```cpp
int main() {
    const int Max_booking = 10;
    FlightBooking booking[Max_booking];

    std::string input = " ";
    while (1)
    {
        int reserved = 0;
        int capacity = 0;
        int seats = 0;
        int ID = 0;
        bool command_worked = false;
        input = " ";

        for (int idx = 0; idx < Max_booking;idx++)
        {
            booking[idx].printStatus();
        }

        std::cout << "What do you want to do?" << std::endl;
        std::cin >> input;

        if (input == "exit") { break; }
        if (input == "add")
        {
            std::cout << "Provide a flight ID: ";
            std::cin >> ID;
            std::cout << "Enter the amount of reservations you wish to add: ";
            std::cin >> seats;
            for (int idx = 0; idx < Max_booking;idx++)
            {
                if (booking[idx].getID() == ID)
                    command_worked = booking[idx].reserveSeats(seats);
            }
        }
        else if (input == "cancel")
        {
            std::cout << "Provide a flight ID: ";
            std::cin >> ID;
            std::cout << "Enter the amount of reservations you wish to cancel: ";
            std::cin >> seats;
            for (int idx = 0; idx < Max_booking;idx++)
            {
                if (booking[idx].getID() == ID)
                {
                    command_worked = booking[idx].cancelReservations(seats);
                }
            }
        }
        else if (input == "create")
        {
            bool Repeat = false;

            while (ID == 0) //Flight cant be 0
            {
                std::cout << "Provide a flight ID (cannot be 0): ";
                std::cin >> ID;
                if (ID == 0) {std::cout << "ID cant be 0";}
            }

            std::cout << "Provide flight capacity: ";
            std::cin >> capacity;
            std::cout << "Provide number of reserved seats: ";
            std::cin >> reserved;


            for (int idx = 0; idx < Max_booking;idx++) //Checking for repeat IDs
            {
                if (booking[idx].getID() == ID)
                {
                    Repeat = true;
                    break;
                }
            }
            if (Repeat)
            {
                std::cout << "IDs can't repeat...";
                continue;
            }

            for (int idx = 0; idx < Max_booking;idx++) //Creating the flight
            {
                if (booking[idx].getID() == 0)
                {
                    booking[idx] = FlightBooking(ID, capacity, reserved);
                    command_worked = true;
                    break;
                }
            }
        }
        else if (input == "delete")
        {
            std::cout << "Provide a flight ID: ";
            std::cin >> ID;


            for (int idx = 0; idx < Max_booking;idx++)
            {
                if (booking[idx].getID() == ID)
                {
                    booking[idx] = FlightBooking();
                    command_worked = true;
                    break;
                }
            }
        }
        else
        {
            std::cout << "Command not found" << std::endl;
            continue;
        }

        if (!command_worked)
        {
            std::cout << "Cannot perform this operation" << std::endl;
        }
    }
    return 0;
```

Output:

```
What do you want to do?
create
Provide a flight ID (cannot be 0): 22
Provide flight capacity: 22
Provide number of reserved seats: 21
Flight [22] : 21/22 (95.4545 %) seats taken
What do you want to do?
create
Provide a flight ID (cannot be 0): 23
Provide flight capacity: 33
Provide number of reserved seats: 39
The reserved seats can not be higher than 105% the plane capacity
Flight [22] : 21/22 (95.4545 %) seats taken
Flight [23] : 0/33 (0 %) seats taken
What do you want to do?
create
Provide a flight ID (cannot be 0): 22
Provide flight capacity: 32
Provide number of reserved seats: 23
IDs can't repeat...Flight [22] : 21/22 (95.4545 %) seats taken
Flight [23] : 0/33 (0 %) seats taken
What do you want to do?
delete
Provide a flight ID: 22
Flight [23] : 0/33 (0 %) seats taken
What do you want to do?
exit

C:\Users\aleja\source\3er Parcial\Laboratorio POO c++\x64\Debug\L
0 (0x0).
Press any key to close this window . . .
```

En este laboratorio vamos paso a paso creando una clase que almacene una ID, la capacidad de un vuelo, y las reservaciones.

El primer objetivo era tener un método que lo imprimiera de la forma “Flight 20 : 1/10 (10%) seats occupied”,

Los siguientes objetivos fueron hacer métodos para reservar y cancelar vuelos, para por último integrarlos con un menú que reconociera diferentes comandos y pudiera: Crear, borrar, reservar asientos a un vuelo con ID específica, cancelar asientos con un vuelo con ID específica y tener en cuenta sus limitaciones (No pueden haber más reservas que 105% la capacidad de un vuelo, un vuelo no puede tener ID de 0, el máximo de vuelos en el sistema son 10, etc).

Laboratorio “1.2.15 - Gym membership management system”

Clase Subscriptions:

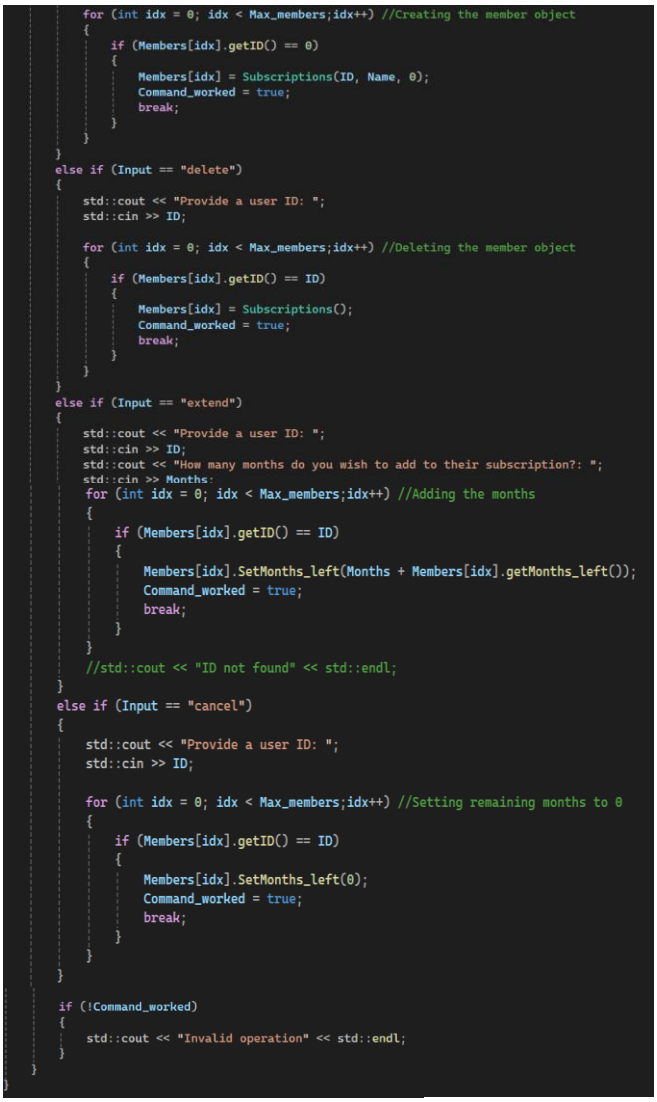
```cpp
#include <iostream>

class Subscriptions
{
private:
    int ID;
    std::string Name;
    int Months_left;
public:
    Subscriptions();
    Subscriptions(int ID, std::string Name, int Months_left);

    int getID() { return ID; }
    int getMonths_left() { return Months_left; }

    void SetMonths_left(int Months_left) { this->Months_left = Months_left; }
    void PrintStatus();
};

Subscriptions::Subscriptions()
{
    this->ID = 0;
    this->Name = " ";
    this->Months_left = 0;
}

Subscriptions::Subscriptions(int ID , std::string Name, int Months_left)
{
    this->ID = ID;
    this->Name = Name;
    this->Months_left = Months_left;
}
void Subscriptions::PrintStatus()
{
    if (ID == 0) { return; }
    std::cout << "User " << ID << " of the name: " << Name << ", has " << Months_left << " months of subscription left." << std::endl;
}
```

Función main():

```cpp
int main()
{
    const int Max_members = 10;
    Subscriptions Members[Max_members];
    std::string Input = " ";

    while (1)
    {
        int ID = 0;
        std::string Name = " ";
        int Months = 0;
        bool Command_worked = false;

        Input = " ";

        std::cout << "\n--- Current Subscriptions ---\n";
        for (int idx = 0; idx < Max_members; idx++) {
            Members[idx].PrintStatus();
        }
        std::cout << "----------------------------\n";

        std::cout << "Welcome to the system, what do you want to do?" << std::endl;
        std::cin >> Input;

        if (Input == "quit") { break;}
if (Input == "create")
{
    bool Repeat = false;

    while (ID == 0) //User ID cant be 0
    {
        std::cout << "Provide a user ID (cannot be 0): ";
        std::cin >> ID;
        if (ID == 0) { std::cout << "ID cant be 0"; }
    }

    std::cout << "Provide the user's name: ";
    std::cin >> Name;


    for (int idx = 0; idx < Max_members;idx++) //Checking for repeat IDs
    {
        if (Members[idx].getID() == ID)
        {
            Repeat = true;
            break;
        }
    }
    if (Repeat)
    {
        std::cout << "IDs can't repeat...";
        continue;
    }
```

Output:

```
--- Current Subscriptions ---
----------------------------
Welcome to the system, what do you want to do?
create
Provide a user ID (cannot be 0): 30
Provide the user's name: Jane

--- Current Subscriptions ---
User 30 of the name: Jane, has 0 months of subscription left.
----------------------------
Welcome to the system, what do you want to do?
extend
Provide a user ID: 30
How many months do you wish to add to their subscription?: 21

--- Current Subscriptions ---
User 30 of the name: Jane, has 21 months of subscription left.
----------------------------
Welcome to the system, what do you want to do?
cancel
Provide a user ID: 30

--- Current Subscriptions ---
User 30 of the name: Jane, has 0 months of subscription left.
----------------------------
Welcome to the system, what do you want to do?
delete
Provide a user ID: 30

--- Current Subscriptions ---
----------------------------
Welcome to the system, what do you want to do?
quit

C:\Users\aleja\source\3er Parcial\Laboratorio POO c++\x64\Debug\
0 (0x0).
Press any key to close this window . . .
```



```
        for (int idx = 0; idx < Max_members;idx++) //Creating the member object
        {
            if (Members[idx].getID() == 0)
            {
                Members[idx] = Subscriptions(ID, Name, 0);
                Command_worked = true;
                break;
            }
        }
    }
    else if (Input == "delete")
    {
        std::cout << "Provide a user ID: ";
        std::cin >> ID;

        for (int idx = 0; idx < Max_members;idx++) //Deleting the member object
        {
            if (Members[idx].getID() == ID)
            {
                Members[idx] = Subscriptions();
                Command_worked = true;
                break;
            }
        }
    }
    else if (Input == "extend")
    {
        std::cout << "Provide a user ID: ";
        std::cin >> ID;
        std::cout << "How many months do you wish to add to their subscription?: ";
        std::cin >> Months;
        for (int idx = 0; idx < Max_members;idx++) //Adding the months
        {
            if (Members[idx].getID() == ID)
            {
                Members[idx].SetMonths_left(Months + Members[idx].getMonths_left());
                Command_worked = true;
                break;
            }
        }
        //std::cout << "ID not found" << std::endl;
    }
    else if (Input == "cancel")
    {
        std::cout << "Provide a user ID: ";
        std::cin >> ID;

        for (int idx = 0; idx < Max_members;idx++) //Setting remaining months to 0
        {
            if (Members[idx].getID() == ID)
            {
                Members[idx].SetMonths_left(0);
                Command_worked = true;
                break;
            }
        }
    }

    if (!Command_worked)
    {
        std::cout << "Invalid operation" << std::endl;
    }
}
}
```

Este laboratorio es un repaso general, el objetivo es, como el anterior laboratorio, crear un sistema para manejar varios objetos solo que esta vez orientado a un sistema de subscripciones a un gym. Estos objetos almacenan lo siguiente:

- Una ID
- Un Nombre
- Los meses restantes de su subscripción

Los métodos que utillizamos son los siguientes:

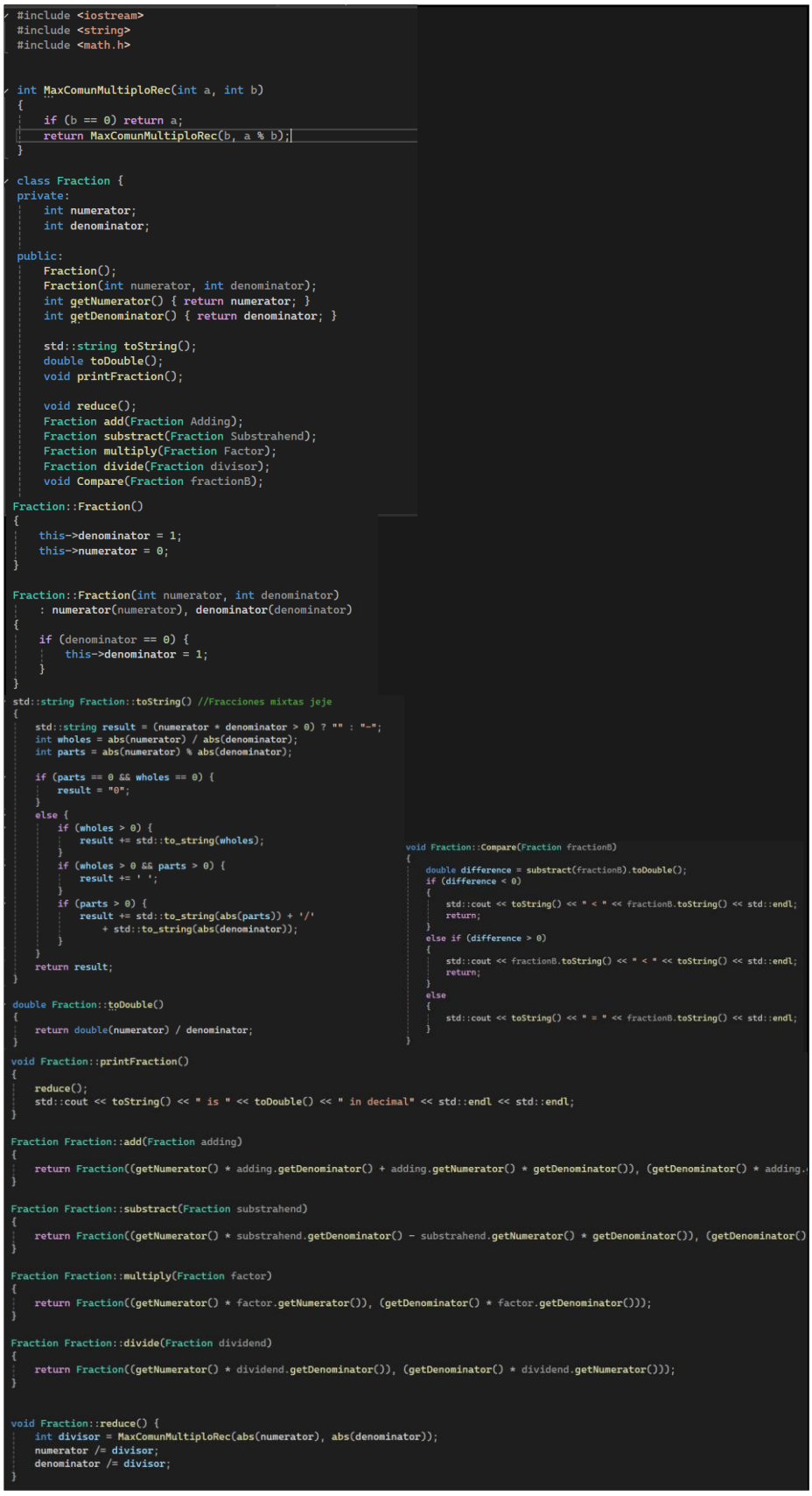
- Algunos getters y setters
- Una función para imprimir los datos de un usuario en un renglón de forma user-friendly

Y en la función main() debemos poder realizar los siguientes comandos:

- **Create** y **delete**: Para añadir un miembro al sistema y asignarle una ID y para borrarlo.
- **Extend** y **Cancel**: Para añadir meses a su subscripción y para dejarlos en 0.
- **Quit**: Para finalizar el programa.

Laboratorio “1.2.16 – 1.2.18 Modelling fractions”

Clase Fraction:



```cpp
#include <iostream>
#include <string>
#include <math.h>

int MaxComunMultiploRec(int a, int b)
{
    if (b == 0) return a;
    return MaxComunMultiploRec(b, a % b);
}

class Fraction {
private:
    int numerator;
    int denominator;

public:
    Fraction();
    Fraction(int numerator, int denominator);
    int getNumerator() { return numerator; }
    int getDenominator() { return denominator; }

    std::string toString();
    double toDouble();
    void printFraction();

    void reduce();
    Fraction add(Fraction Adding);
    Fraction substract(Fraction Substrahend);
    Fraction multiply(Fraction Factor);
    Fraction divide(Fraction divisor);
    void Compare(Fraction fractionB);
Fraction::Fraction()
{
    this->denominator = 1;
    this->numerator = 0;
}

Fraction::Fraction(int numerator, int denominator)
    : numerator(numerator), denominator(denominator)
{
    if (denominator == 0) {
        this->denominator = 1;
    }
}
std::string Fraction::toString() //Fracciones mixtas jeje
{
    std::string result = (numerator * denominator > 0) ? "" : "-";
    int wholes = abs(numerator) / abs(denominator);
    int parts = abs(numerator) % abs(denominator);

    if (parts == 0 && wholes == 0) {
        result = "0";
    }
    else {
        if (wholes > 0) {
            result += std::to_string(wholes);
        }
        if (wholes > 0 && parts > 0) {
            result += ' ';
        }
        if (parts > 0) {
            result += std::to_string(abs(parts)) + '/'
                + std::to_string(abs(denominator));
        }
    }
    return result;
}

double Fraction::toDouble()
{
    return double(numerator) / denominator;
}
void Fraction::Compare(Fraction fractionB)
{
    double difference = substract(fractionB).toDouble();
    if (difference < 0)
    {
        std::cout << toString() << " < " << fractionB.toString() << std::endl;
        return;
    }
    else if (difference > 0)
    {
        std::cout << fractionB.toString() << " < " << toString() << std::endl;
        return;
    }
    else
    {
        std::cout << toString() << " = " << fractionB.toString() << std::endl;
    }
}
void Fraction::printFraction()
{
    reduce();
    std::cout << toString() << " is " << toDouble() << " in decimal" << std::endl << std::endl;
}

Fraction Fraction::add(Fraction adding)
{
    return Fraction((getNumerator() * adding.getDenominator() + adding.getNumerator() * getDenominator()), (getDenominator() * adding.
}

Fraction Fraction::substract(Fraction substrahend)
{
    return Fraction((getNumerator() * substrahend.getDenominator() - substrahend.getNumerator() * getDenominator()), (getDenominator()
}

Fraction Fraction::multiply(Fraction factor)
{
    return Fraction((getNumerator() * factor.getNumerator()), (getDenominator() * factor.getDenominator()));
}

Fraction Fraction::divide(Fraction dividend)
{
    return Fraction((getNumerator() * dividend.getDenominator()), (getDenominator() * dividend.getNumerator()));
}


void Fraction::reduce() {
    int divisor = MaxComunMultiploRec(abs(numerator), abs(denominator));
    numerator /= divisor;
    denominator /= divisor;
}
```

Función Main:

```cpp
int main() {
    Fraction fraction[2];
    int numerator, denominator;
    char slash;
    std::string input;

    for (int i = 0;i < 2;i++)
    {
        std::cout << "Enter a fraction (for example : 3 / 4): ";
        if (std::cin >> numerator >> slash >> denominator && slash == '/')
        {
            fraction[i] = Fraction(numerator, denominator);
            fraction[i].printFraction();
        }
        else {
            std::cout << "Unrecognized input format, try something more like : '3 / 4' pls" << std::endl;
        }
    }

    std::cout << fraction[0].toString() << " plus " << fraction[1].toString() << std::endl;
    (fraction[0].add(fraction[1])).printFraction();

    std::cout << fraction[0].toString() << " minus " << fraction[1].toString() << std::endl;
    (fraction[0].substract(fraction[1])).printFraction();

    std::cout << fraction[0].toString() << " times " << fraction[1].toString() << std::endl;
    (fraction[0].multiply(fraction[1])).printFraction();

    std::cout << fraction[0].toString() << " by " << fraction[1].toString() << std::endl;
    (fraction[0].divide(fraction[1])).printFraction();

    fraction[0].Compare(fraction[1]);
```

Output:

```
Enter a fraction (for example : 3 / 4): 6/4
1 1/2 is 1.5 in decimal

Enter a fraction (for example : 3 / 4): 4/6
2/3 is 0.666667 in decimal

1 1/2 plus 2/3
2 1/6 is 2.16667 in decimal

1 1/2 minus 2/3
5/6 is 0.833333 in decimal

1 1/2 times 2/3
1 is 1 in decimal

1 1/2 by 2/3
2 1/4 is 2.25 in decimal

2/3 < 1 1/2

C:\Users\aleja\source\3er Parcial\Laboratorio
0 (0x0).
Press any key to close this window . . .
```

En este laboratorio creamos una clase que represente fracciones, esta debe de almacenar un numerador y denominador entero (el denominador no puede ser 0). Esta clase debe tener métodos para:

- Métodos Sumar, restar, multiplicar y dividir fracciones. (**add**, **substract**, **multiply**, **divide**)
- Devolver la fracción en string, las fracciones deben ser mixtas en este caso (**toString**).
- Devolver la fracción en double (**toDouble**).
- Comparar fracciones (**Compare**).

En la función **main**() pedimos dos fracciones para realizar una demostración de todos los métodos de la clase.

Laboratorio “1.2.19 – 1.2.21 Points in 2D”

Clase Point2D:

```cpp
#include <iostream>
#include <string>
#include <math.h>


class Point2D {
public:
    Point2D(double x, double y);
    Point2D();
    std::string toString();

    double getX() { return x; }
    double getY() { return y; }
    void setX(double x) { this->x = x; }
    void setY(double y) { this->y = y; }

    double distanceTo(Point2D that);
private:
    double x;
    double y;
};


Point2D::Point2D()
{
    this->x = 0;
    this->y = 0;
}

Point2D::Point2D(double x, double y)
{
    this->x = x;
    this->y = y;
}
std::string Point2D::toString()
{
    std::string Wasa = std::to_string(x) + " , " + std::to_string(y);
    return Wasa;
}

double Point2D::distanceTo(Point2D that)
{
    double diffX = x - that.getX();
    if (diffX < 0) { diffX *= (-1); }

    double diffY = y - that.getY();
    if (diffY < 0) { diffY *= (-1); }

    return sqrt(pow(diffX, 2) + pow(diffY, 2));
}
```

Clase Line2D:

```cpp
class Line2D {
public:
    Line2D(Point2D, Point2D);
    double getSlope() { return slope; }
    double getYIntersect() { return yIntersect; }
    Point2D getPointA() { return pointA; }
    Point2D getPointB() { return pointB; }
    std::string toString();


private:
    Point2D pointA;
    Point2D pointB;
    double slope;
    double yIntersect;
};

Line2D::Line2D(Point2D pointA, Point2D pointB)
{
    this->pointA = pointA;
    this->pointB = pointB;
    slope = ((pointB.getY() - pointA.getY()) / (pointB.getX() - pointA.getX()));
    yIntersect = (pointA.getY() - slope * pointA.getX());
}

std::string Line2D::toString() {
    return "Line2D -> PointA(" + pointA.toString() +
        "), PointB(" + getPointB().toString() +
        "), slope = " + std::to_string(slope) +
        ", yIntersect = " + std::to_string(yIntersect);
}
```

Función Main();

```cpp
int main() {
    Point2D Points[3];
    std::string input = " ";
    double pointX;
    double pointY;
    char comma;

    while (input != "exit")
    {
        for (int i = 0; i < 2;i++) {
            std::cout << "Please provide a point in (3,4) format: ";
            std::cin >> pointX >> comma >> pointY;

            if (comma == ',')
            {
                Points[i] = Point2D(pointX, pointY);
            }

            std::cout << "Added point " << Points[i].toString() << std::endl;
        }

        Line2D LineAB(Points[0], Points[1]);
        std::cout << "The distance between point " << Points[0].toString() << " and point " << Points[1].toString() << " is " << Point

        std::cout << "Line ecuation: Y = " << LineAB.getSlope() << "x +" << LineAB.getYIntersect() << std::endl;

        std::cout << LineAB.toString() << std::endl << std::endl;

        std::cout << "Enter a C point to check colinearity:";
        std::cin >> pointX >> comma >> pointY;
         if (comma == ',')
         {
             Points[2] = Point2D(pointX,pointY);
         }

         if (Points[2].getY() == (LineAB.getSlope() * Points[2].getX()) + LineAB.getYIntersect())
         {
             std::cout << "Point C is colinear with Point A and Point B" << std::endl;
         }else
         {
             std::cout << "Point C isnt colinear with Point A and Point B" << std::endl;
         }

    }
}
```

Output:

```
Please provide a point in (3,4) format: 5,6
Added point 5.000000 , 6.000000
Please provide a point in (3,4) format: 7,8
Added point 7.000000 , 8.000000
The distance between point 5.000000 , 6.000000 and point 7.000000 , 8.000000 is 2.82843
Line ecuation: Y = 1x +1
Line2D -> PointA(5.000000 , 6.000000), PointB(7.000000 , 8.000000), slope = 1.000000, yIntersect = 1.000000

Enter a C point to check colinearity:1313,1314
Point C is colinear with Point A and Point B

C:\Users\aleja\source\3er Parcial\Laboratorio POO c++\x64\Debug\Laboratorio 1-2-21.exe (process 25488) exite
0 (0x0).
Press any key to close this window . . .
```

En este laboratorio creamos una clase Punto que guarde un valor X y Y, esta tiene métodos para conseguir la distancia con otro punto mediante el teorema de Pitágoras con getters y setters, etc. Después, creamos una clase Line2D que calcule la pendiente e intersección con 2 puntos, y por último un método que determine si un punto es colinear con la ecuación de la línea (si cumple Y = mX + b).

Laboratorio “1.2.22 – 1.2.23 Inheritance basics”

Clase FarmAnimal y sus subclases:

```cpp
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

class FarmAnimal {
public:
    FarmAnimal(double water_consumption);
    double getWaterConsumption();
private:
    double water_consumption;
};

FarmAnimal::FarmAnimal(double water_consumption) {
    this->water_consumption = water_consumption;
}

double FarmAnimal::getWaterConsumption() {return water_consumption;}

// ---- Derived animal classes ----
class Sheep : public FarmAnimal {
public:
    Sheep(double weight)
        : FarmAnimal(1.1 * (weight / 10.0)) {}
};

class Horse : public FarmAnimal {
public:
    Horse(double weight)
        : FarmAnimal(6.8 * (weight / 100.0)) {}
};
class Cow : public FarmAnimal {
public:
    Cow(double weight)
        : FarmAnimal(8.6 * (weight / 100.0)) {}
};
```

Clase Consumption Accumulator:

```cpp
// ---- ConsumptionAccumulator ----
class ConsumptionAccumulator {
public:
    ConsumptionAccumulator();
    double getTotalConsumption();
    void addConsumption(FarmAnimal& animal);
private:
    double total_consumption;
};

ConsumptionAccumulator::ConsumptionAccumulator() : total_consumption(0) {}

double ConsumptionAccumulator::getTotalConsumption() {
    return total_consumption;
}

void ConsumptionAccumulator::addConsumption(FarmAnimal& animal) {
    total_consumption += animal.getWaterConsumption();
}
```

Función Main:

```cpp
int main() {
    ConsumptionAccumulator accumulator;
    string line;

    while (true) {
        std::cout << "Enter your animal and its weight, for example: sheep 50" << endl;
        getline(cin, line);
        if (line.empty()) break; //Menú con getline que para al dejar una linea vacía

        stringstream ss(line);
        string type;
        double weight;

        ss >> type >> weight;
        if (type == "end") { break; }
        if (type == "sheep") {
            Sheep s(weight);
            accumulator.addConsumption(s);
            cout << "Your sheep will consume " << s.getWaterConsumption() << " liters of water per day" << endl;
        }
        else if (type == "horse") {
            Horse h(weight);
            accumulator.addConsumption(h);
            cout << "Your horse will consume " << h.getWaterConsumption() << " liters of water per day" << endl;
        }
        else if (type == "cow") {
            Cow c(weight);
            accumulator.addConsumption(c);
            cout << "Your cow will consume " << c.getWaterConsumption() << " liters of water per day" << endl;
        }
        else {
            cout << "Unknown animal: " << type << endl;
        }
        else {
            cout << "Unknown animal: " << type << endl;
        }

        cout << "Your total liter consumption per day is: " << accumulator.getTotalConsumption() << endl << endl;
    }

    return 0;
}
```

En este laboratorio el objetivo es entender las subclases creando subclases de la clase **FarmAnimal** que calculen el consumo de agua de diferentes animales de forma diferente para luego hacer una sumatoria en una clase **ConsumptionAccumulator** que guardará el total hasta el momento. Creamos subclases **Sheep**, **Horse** y **Cow** que tendrán definidos el método **getWaterConsumption()** de forma diferente. La función **main** añade el consumo de los animales que inserte el usuario de forma “(animal) (kilos)” e imprime el total al final de la ejecución.

Output:

```
Enter your animal and its weight, for example: sheep 50
sheep 33
Your sheep will consume 3.63 liters of water per day
Your total liter consumption per day is: 3.63

Enter your animal and its weight, for example: sheep 50
horse 30
Your horse will consume 2.04 liters of water per day
Your total liter consumption per day is: 5.67

Enter your animal and its weight, for example: sheep 50
cow 222
Your cow will consume 19.092 liters of water per day
Your total liter consumption per day is: 24.762

Enter your animal and its weight, for example: sheep 50
exit
Unknown animal: exit
Your total liter consumption per day is: 24.762

Enter your animal and its weight, for example: sheep 50
end

C:\Users\aleja\source\3er Parcial\Laboratorio POO c++\x64
0 (0x0).
Press any key to close this window . . .
```

Laboratorio “1.2.24 - 1.2.30 Singly linked list”

Clase Node y Clase List:

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    Node(int val);
    ~Node();
    int value;
    Node* next;
};

Node::Node(int val) : value(val), next(nullptr) {}
Node::~Node() {}

class List {
public:
    List();
    List(const List& copy);
    ~List();
    int size();
    void push_front(int value);
    void push_back(int value);
    bool pop_front(int& value);
    bool pop_back(int& value);
    int at(int index);
    void insert_at(int index, int value);
    void remove_at(int index);
private:
    Node* head;
    Node* tail;
};

List::List() : head(nullptr), tail(nullptr) {}

List::List(const List& copy) : head(nullptr), tail(nullptr) /
{
    Node* curr = copy.head;
    while (curr != nullptr) {
        //Y aquí se inicializan
        push_back(curr->value);
        curr = curr->next;
    }
}

List::~List() {
    while (head != nullptr) {
        Node* n = head;
        head = head->next;
        delete n;
    }
}

int List::size() {
    int totalSize{ 0 };
    Node* reco = head;
    while (reco != nullptr) {
        totalSize += 1;
        reco = reco->next;
    }
    return totalSize;
}

void List::push_front(int value) {
    Node* new_head = new Node(value);
    new_head->next = head;
    head = new_head;
    if (head->next == nullptr) {
        tail = head;
    }
}

void List::push_back(int value) {
    if (head == nullptr) {
        push_front(value);
        return;
    }
    Node* new_tail = new Node(value);
    tail->next = new_tail;
    tail = new_tail;
}

bool List::pop_front(int& value) {
    if (head != nullptr) {
        Node* popped = head;
        head = head->next;
        value = popped->value;
        delete popped;
        return true;
    }
    return false;
}

bool List::pop_back(int& value) {
    if (tail != nullptr) {
        if (head == tail) {
            value = head->value;
            delete head;
            head = tail = nullptr;
            return true;
        }
        Node* new_tail = head;
        while (new_tail->next != tail) {
            new_tail = new_tail->next;
        }
        value = tail->value;
        delete tail;
        tail = new_tail;
        tail->next = nullptr;
        return true;
    }
    // implement the pop
    // don't forget to delete the popped node!
    // and fix the return value
    return false;
}
```

```cpp
int List::at(int index) {
    Node* Reco = head;
    for (int i = 0;i < index || index < 0;i++) {
        Reco = Reco->next;
        if (Reco == nullptr) {
            break;
        }
    }
    if (Reco == nullptr) {
        cout << "Invalid index" << endl;return 0;
    }
    return Reco->value;
}
void List::remove_at(int index) {
    if (index >= size() || index < 0) {
        cout << "Invalid index" << endl;return;
    }
    Node* DeletedNode = head;
    Node* DeletedNodePrev = head;
    if (index == 0) {
        //Para indice 0 y listas de un solo elemento
        head = head->next;
        if (head == nullptr) {
            tail = nullptr;
        }
        delete DeletedNode;
        return;
    }
    for (int i = 0;i < index;i++) {
        DeletedNodePrev = DeletedNode;
        DeletedNode = DeletedNode->next;
    }
    if (DeletedNode == tail) {
        DeletedNodePrev->next = nullptr;
        tail = DeletedNodePrev;
    }
    else {
        DeletedNodePrev->next = DeletedNode->next;
    }
    delete DeletedNode;
}
void List::insert_at(int index, int value) {
    if (index > size() || index < 0) {
        cout << "Invalid index" << endl;return;
    }
    else if (index == 0) {
        push_front(value);
        return;
    }
    else if (index == size()) {
        push_back(value);
        return;
    }
    Node* RecoPrev = head;
    Node* NewNode = new Node(value);
    for (int i = 0;i < index - 1;i++) {
        RecoPrev = RecoPrev->next;
    }
    NewNode->next = RecoPrev->next;
    RecoPrev->next = NewNode;
}

void printList(List& list) {
    for (int i = 0; i < list.size(); i++) {
        cout << "list[" << i << "] == " << list.at(i) << endl;
    }
}
```

Función Main()

```cpp
int main()
{
    List list1;
    int value;
    list1.push_front(1);
    list1.push_back(2);
    list1.push_front(3);
    list1.push_back(4);
    list1.push_front(5);
    list1.push_back(6);

    cout << "list1" << endl;
    if (list1.pop_back(value)) { std::cout << "Popped " << value << endl; }
    if (list1.pop_front(value)) { std::cout << "Popped " << value << endl; }
    printList(list1);
    cout << endl;

    List list2(list1);
    cout << "list2" << endl;
    printList(list2);
    cout << endl;

    list1.insert_at(1, 6);
    list2.remove_at(2);

    cout << "list1" << endl;
    printList(list1);
    cout << "list2" << endl;
    printList(list2);
    cout << endl;

    return 0;
}
```

Output

```
Popped 6
Popped 5
list1
list[0] == 3
list[1] == 1
list[2] == 2
list[3] == 4

list2
list[0] == 3
list[1] == 1
list[2] == 2
list[3] == 4

list1
list[0] == 3
list[1] == 6
list[2] == 1
list[3] == 2
list[4] == 4

list2
list[0] == 3
list[1] == 1
list[2] == 4


C:\Users\aleja\source\3er Parcial\Laborat
0 (0x0).
Press any key to close this window . . .
```

En este laboratorio implementamos una lista enlazada simple mediante dos clases: **Node**, que representa un nodo con un valor entero y un puntero al siguiente, y **List**, que administra la lista completa. Los métodos de List permiten construir la lista, copiarla, y destruirla liberando memoria,ntre mientras que en sus métodos destacan: **push_front** y **push_back** para insertar nodos al inicio o final; pop_front y pop_back para extraer y eliminar nodos devolviendo su valor; **size** para calcular la cantidad de elementos; at para acceder a un valor en un índice; **insert_at** y **remove_at** para insertar o eliminar en una posición específica. El programa principal prueba estas operaciones creando una lista, realizando inserciones y eliminaciones, mostrando los resultados, y comprobando el correcto funcionamiento de la copia de listas.

Laboratorio “1.2.24 - 1.2.30 Singly linked list”

Clase Node y Clase List:

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    Node(int val);
    ~Node();
    int value;
    Node* next;
    Node* prev;
};

Node::Node(int val) : value(val), next(nullptr), prev(nullptr) {}
Node::~Node() {}

class List {
public:
    List();
    List(const List& copy);
    ~List();
    int size() const;
    void push_front(int value);
    void push_back(int value);
    bool pop_front(int& value);
    bool pop_back(int& value);

    int at(int index) const;
    void insert_at(int index, int value);
    void remove_at(int index);

private:
    Node* head;
    Node* tail;
};

List::List(const List& copy) : head(nullptr), tail(nullptr) {
    Node* curr = copy.head;
    while (curr != nullptr) {
        push_back(curr->value);
        curr = curr->next;
    }
}

List::~List() {
    while (head != nullptr) {
        Node* n = head;
        head = head->next;
        delete n;
    }
}

int List::size() const {
    int totalSize = 0;
    Node* curr = head;
    while (curr != nullptr) {
        totalSize++;
        curr = curr->next;
    }
    return totalSize;
}
void List::push_front(int value) {
    Node* new_head = new Node(value);
    new_head->next = head;
    if (head != nullptr) {
        head->prev = new_head;
    }
    else {
        tail = new_head; // lista estaba vacía
    }
    head = new_head;
}

void List::push_back(int value) {
    Node* new_tail = new Node(value);
    if (tail != nullptr) {
        tail->next = new_tail;
        new_tail->prev = tail;
        tail = new_tail;
    }
    else {
        head = tail = new_tail;
    }
}
```

```cpp
bool List::pop_front(int& value) {
    if (head == nullptr) return false;
    Node* popped = head;
    value = popped->value;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    }
    else {
        tail = nullptr; // la lista quedó vacía
    }
    delete popped;
    return true;
}

bool List::pop_back(int& value) {
    if (tail == nullptr) return false;
    Node* popped = tail;
    value = popped->value;
    tail = tail->prev;
    if (tail != nullptr) {
        tail->next = nullptr;
    }
    else {
        head = nullptr; // lista quedó vacía
    }
    delete popped;
    return true;
}
int List::at(int index) const {
    if (index < 0 || index >= size()) {
        cout << "Invalid index" << endl;
        return 0;
    }
    Node* curr = head;
    for (int i = 0; i < index; i++) {
        curr = curr->next;
    }
    return curr->value;
}
void List::insert_at(int index, int value) {
    if (index < 0 || index > size()) {
        cout << "Invalid index" << endl;
        return;
    }
    if (index == 0) {
        push_front(value);
        return;
    }
    if (index == size()) {
        push_back(value);
        return;
    }

    Node* curr = head;
    for (int i = 0; i < index; i++) {
        curr = curr->next;
    }

    Node* new_node = new Node(value);
    Node* prev_node = curr->prev;

    new_node->next = curr;
    new_node->prev = prev_node;
    prev_node->next = new_node;
    curr->prev = new_node;
}
void List::remove_at(int index) {
    if (index < 0 || index >= size()) {
        cout << "Invalid index" << endl;
        return;
    }
    if (index == 0) {
        int dummy;
        pop_front(dummy);
        return;
    }
    if (index == size() - 1) {
        int dummy;
        pop_back(dummy);
        return;
    }

    Node* curr = head;
    for (int i = 0; i < index; i++) {
        curr = curr->next;
    }

    Node* prev_node = curr->prev;
    Node* next_node = curr->next;

    prev_node->next = next_node;
    next_node->prev = prev_node;

    delete curr;
}
void printList(const List& list) {
    for (int i = 0; i < list.size(); i++) {
        cout << "list[" << i << "] == " << list.at(i) << endl;
    }
}
```

Función Main()

```cpp
int main()
{
    List list1;
    int value;
    list1.push_front(1);
    list1.push_back(2);
    list1.push_front(3);
    list1.push_back(4);
    list1.push_front(5);
    list1.push_back(6);

    cout << "list1" << endl;
    if (list1.pop_back(value)) { std::cout << "Popped " << value << endl; }
    if (list1.pop_front(value)) { std::cout << "Popped " << value << endl; }
    printList(list1);
    cout << endl;

    List list2(list1);
    cout << "list2" << endl;
    printList(list2);
    cout << endl;

    list1.insert_at(1, 6);
    list2.remove_at(2);

    cout << "list1" << endl;
    printList(list1);
    cout << "list2" << endl;
    printList(list2);
    cout << endl;

    return 0;
}
```

Output

```
Popped 6
Popped 5
list1
list[0] == 3
list[1] == 1
list[2] == 2
list[3] == 4

list2
list[0] == 3
list[1] == 1
list[2] == 2
list[3] == 4

list1
list[0] == 3
list[1] == 6
list[2] == 1
list[3] == 2
list[4] == 4

list2
list[0] == 3
list[1] == 1
list[2] == 4


C:\Users\aleja\source\3er Parcial\Laborat
0 (0x0).
Press any key to close this window . . .
```

En este laboratorio implementamos una lista enlazada doble mediante dos clases: **Node**, que representa un nodo con un valor entero, un puntero al siguiente y un puntera al anteriior, y **List**, que administra la lista completa. Los métodos de List permiten construir la lista, copiarla, y destruirla liberando memoria,ntre mientras que en sus métodos destacan: **push_front** y **push_back** para insertar nodos al inicio o final; pop_front y pop_back para extraer y eliminar nodos devolviendo su valor; **size** para calcular la cantidad de elementos; at para acceder a un valor en un índice; **insert_at** y **remove_at** para insertar o eliminar en una posición específica. El programa principal prueba estas operaciones creando una lista, realizando inserciones y eliminaciones, mostrando los resultados, y comprobando el correcto funcionamiento de la copia de listas. La diferencia principal al anterior es el comportamiento de los métodos debido al puntero que apunta al nodo anterior.