UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE CIENCIAS DE LA COMPUTACIÓN

LABORATORIO 19 – Functores.

DOCENTE:

Enzo Edir Velásquez Lobatón

ALUMNO:

Owen Haziel Roque Sosa.

FECHA:

10/08/2022

Arequipa – Perú

1. Utilizando Functores, elabore una clase que calcule la ecuación de regresión lineal simple (y = a + bc) de un conjunto de pares de datos (x,y) almacenados en un vector, retorne como parte del resultado los valores de a y b. Apóyese en functores para calcular las diferentes sumatorias que pueden presentarse.

Se utilizó el patrón de diseño Strategy para implementar los functores:

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

class AbstractSumatorias {
public:
    virtual ~AbstractSumatorias() {};
    virtual int operator()(const vector<pair<int,int>>& data) const = 0;
};

class Context {
private:
    AbstractSumatorias *ptr_sumatoria;
public:
    Context(AbstractSumatorias *ptr = nullptr) : ptr_sumatoria(ptr) {}
    ~Context(){ delete this->ptr_sumatoria;}
    void setSumatoria(AbstractSumatorias *sumatoria){
        delete this->ptr_sumatoria;
        this->ptr_sumatoria = sumatoria;
    }
    int ejecutarSumatoria(const vector<pair<int,int>>& data) {
        int result = this->ptr_sumatoria->operator()(data);
        return result;
    }
};
```

```cpp
class Sumatoria_x : public AbstractSumatorias {
public:
    int operator()(const vector<pair<int,int>>& v) const override {
        int Sumatoria = 0;
        for(std::size_t i = 0; i < v.size(); i++)
            Sumatoria += v[i].first;
        return Sumatoria;
    }
};

class Sumatoria_y : public AbstractSumatorias {
public:
    int operator()(const vector<pair<int,int>>& v) const override {
        int Sumatoria = 0;
        for(std::size_t i = 0; i < v.size(); i++)
            Sumatoria += v[i].second;
        return Sumatoria;
    }
};

class Sumatoria_xy : public AbstractSumatorias {
public:
    int operator()(const vector<pair<int,int>>& v) const override {
        int Sumatoria = 0;
        for(std::size_t i = 0; i < v.size(); i++){
            int multiply = v[i].first * v[i].second;
            Sumatoria += multiply;
        }
        return Sumatoria;
    }
};
```

```cpp
class Sumatoria_x_square : public AbstractSumatorias {
public:
    int operator()(const vector<pair<int,int>>& v) const override {
        int Sumatoria = 0;
        for(std::size_t i = 0; i < v.size(); i++)
            Sumatoria += pow(v[i].first, 2);
        return Sumatoria;
    }
};

// https://youtu.be/4PiiSUxcalg
class RegLinSim {
// y = a + bx
public:
    double calcularParamA(int n, int sx, int sy, int sxy, int sxx){
        double a = ((n * sxy) - (sx * sy))/((n * sxx) - pow(sx, 2));
        return a;
    }
    double calcularParamB(int n, int sx, int sy, int sxy, int sxx){
        double a = this->calcularParamA(n, sx, sy, sxy, sxx);
        double b = double(sy)/n - a * double(sx)/n;
        return b;
    }
};
```

```cpp
int main(int argc, char *argv[]) {
    // <x, y>
    // https://youtu.be/4PiiSUxcalg
    vector<std::pair<int,int>> vector_pares = {
        {1,2},
        {2,3},
        {2,4},
        {3,4},
        {4,4},
        {4,6},
        {5,5},
        {6,7}
    };
    int n, sum_x, sum_y, sum_xy, sum_x_square;
    // Obtencion de sumatorias a traves de
    // functores y Patron de Disenho Strategy
    n = vector_pares.size();
    //cout << "n: " << n << endl;
    Context *context = new Context(new Sumatoria_x);
    sum_x = context->ejecutarSumatoria(vector_pares);
    //cout << "sx: " << sum_x << endl;
    context->setSumatoria(new Sumatoria_y);
    sum_y = context->ejecutarSumatoria(vector_pares);
    //cout << "sy: " << sum_y << endl;
    context->setSumatoria(new Sumatoria_xy);
    sum_xy = context->ejecutarSumatoria(vector_pares);
    //cout << "sxy: " << sum_xy << endl;
    context->setSumatoria(new Sumatoria_x_square);
    sum_x_square = context->ejecutarSumatoria(vector_pares);
    //cout << "sxx: " << sum_x_square << endl;
    delete context;
    // Ecuacion de regresion lineal simple
    RegLinSim ec;
    cout << "y = a + bx" << endl;
    double param_a = ec.calcularParamA(n, sum_x, sum_y, sum_xy, sum_x_square);
    cout << "Parametro A: " << param_a << endl;
    double param_b = ec.calcularParamB(n, sum_x, sum_y, sum_xy, sum_x_square);
    cout << "Parametro B: " << param_b << endl;
    return 0;
}
```

Video de donde se recuperó las formulas y la data: https://youtu.be/4PiiSUxcalg

```
y = a + bx
Parametro A: 0.849057
Parametro B: 1.50943
```

2. Utilizando Functores, elabore una clase que simule el proceso de la función estándar FIND. Se debe trabajar enviando como parámetros el índice de inicio, el índice final de la búsqueda y el dato a buscar. Retorne todas las ocurrencias iguales dentro del rango indicado (debe devolver un vector con los índices de todas las ocurrencias)

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

class SearchFunct {
public:
    void operator()(vector<int>::iterator ib, vector<int>::iterator ie, int x) {
        vector<int> res;
        unsigned int idx=0;
        while (ib != ie) {
            if (*ib == x)
                res.push_back(idx);
            idx++;
            ++ib;
        }
        if (res.empty())
            cout << "No se encontro el valor en el vector.\n";
        else  {
            for (unsigned int i=0; i<res.size(); i++){
                cout << x << " encontrado en posicion ";
                cout << res.at(i) << "\n";
            }
            cout << endl;
        }
    }
};
```

```cpp
int main(int argc, char *argv[]) {
    vector<int> searchList = {
        10, 12, 16, 81, 32, 54, 78,
        56, 9, 45, 78, 90, 101, 33,
        21, 37, 49, 98, 67, 12, 89,
        78, 12, 90, 24, 59, 49, 56
    };
    SearchFunct searchVector;
    cout << "Busqueda en toda la lista: Valor 90" << endl;
    searchVector(searchList.begin(), searchList.end(), 90);
    cout << endl;
    cout << "Busqueda dentro de un rango: [1 - 10] Valor 90" << endl;
    searchVector(searchList.begin()+1, searchList.begin()+10, 90);
    return 0;
}
```

```
Busqueda en toda la lista: Valor 90
90 encontrado en posicion 11
90 encontrado en posicion 23


Busqueda dentro de un rango: [1 - 10] Valor 90
No se encontro el valor en el vector.
```

3. Utilizando Functores y el método std::sort, elabore una clase Elementos con dos atributos de números enteros a y b. Genere una lista de 20 elementos de esta clase Elementos con valores aleatorios tanto para a y b. Mediante el método std::sort ordénelos de la forma en que un Objeto al ser comparado con un segundo se tenga la desigualdad : obj1.a < obj2.b . El método std::sort debe de trabajar con un objeto Functores. De ser necesario, investigue como realizar este procedimiento que dependa de un objeto del tipo Functores.

```cpp
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <algorithm>
#include <vector>
using namespace std;

class Elementos {
public:
    int a;
    int b;
    Elementos (int _a, int _b) : a(_a), b(_b) {}
};

struct SortV {
    friend class Elementos;
    bool operator()(Elementos *e) { return (e->a < e->b); }
} sortFunct;

int main(int argc, char *argv[]) {
    vector<Elementos> vectorElem = {};
    srand(time(NULL));
    for (int i = 0; i < 20; i++){
        vectorElem.push_back(Elementos((0 + rand() % (51 - 1)), (0 + rand() % (51 - 1))));
        cout << vectorElem[i].a << "    ";
        cout << vectorElem[i].b << endl;
    }
    std::sort(vectorElem.begin(), vectorElem.end(), sortFunct);

    return 0;
}
```

C++ No permite comparar un OBJETO con OTRO OBJETO en el método std::sort, solo vectores. (en ese caso si se puede realizar con functor)