**Trabajo Práctico Nº 8**

1. A diferencia de la programación estructurada en la que poníamos todo en un mismo archivo para resolver algo, con la programación modular separamos en varios archivos que cada uno resuelve una pequeña parte del problema, se pueden testear errores de cada una de estas partes sin comprometer todo el proyecto, además que esas pequeñas partes individuales se pueden usar para otros proyectos que puedan necesitar. Con compilación separada se refiere a que primero se compilan cada una de las partes a código objeto y una vez ejecutado el principal, se adjuntan todos y se forma el ejecutable.
2. La interfaz es todo con lo que el usuario interactúa y como se usan los subprogramas que tenemos separados en los distintos modulos. La implementación es como se hace todo, como funciona todo debajo de la pantalla. Esto es lo que se encuentra dentro de los subprogramas que se llaman en el main y a lo que el usuario no tiene acceso.
3. Los modulos de biblioteca estan compuestos por todo el código y subprogramas y creamos nosotros, es como un stock de soluciones que podemos llamar cuando las necesitamos, tenemos que resolver una suma? Llamamos a uno que resuelve sumas, tenemos que resolver el área de un triangulo? Llamamos a la que calcula el área de un triangulo. Los archivos.h son nuestros archivos de cabecera en donde estan todos los tipos definidos por el usuario y nuestro archivo de implementacion archivos.cpp es lo que utiliza las funciones y subprogramas para resolver nuestros problemas.
4. Con la consola podemos usar la opción g++ para compilar todos los archivos, solo se compilan los .cpp.  
   Archivos de cabecera e implementación en la misma carpeta,listamos todos los .cpp en la orden g++ :E:\AyEDI\Uni08>g++ -o Main.exe Main.cpp Myfunctions.cpp.
5. El preprocesado son un conjunto de directivas que el compilador va interpretando para generar un archivo temporal con todo el código del programa.  
   por ejemplo la directiva #include que si yo tengo dos archivos y uno tiene una lista y otro un #include <string> va a generar un solo archivo con la lista y va a tomar lo que esta dentro de la librería string y lo va a agregar para usarse en el archivo.
6. La programación modular, como su nombre lo dice, tenemos todo separado en modulos diferentes que cada uno se encarga de resolver una pequeña tarea. Esto nos simplifica la lectura del programa para buscar algo, si tenemos un error no tenemos que buscar un error entre un archivo de 1000 lineas de código, los modulos son intercambiables y podemos utilizarlos en otros programas que tengan las mismas problemáticas.
7. El problema de las inclusiones múltiples es que a veces varios archivos incluyen dentro un mismo .h para funcionar correctamente y eso nos va a generar un problema de compilación ya que estamos tratando de llamar muchas veces un archivo exactamente igual.
8. La compilación condicional es un conjunto de directivas (*#ifdef, #ifndef, #else y #endif)*Se usan junto con la directiva #define para decirle al programa que se va a compilar el código if pero en caso de que no,se compilara el código else. Nosotros ponemos las condiciones de que código se compila y que no.
9. Gracias a la compilación condicional y sus directivas (*#ifdef, #ifndef, #else y #endif)*que vamos a usar para poner condiciones a nuestra compilación, entonces cuando nosotros estemos compilando todas nuestras bibliotecas juntas si una ya fue llamada y usada, en caso de que se repita, no la vuelva a llamar y no se generen inclusiones multiples.
10. Los espacios de nombres se utilizan para organizar el código en grupos lógicos y para evitar conflictos de nombres que pueden producirse, especialmente cuando la base de código incluye varias bibliotecas. Todos los identificadores del ámbito del espacio de nombres son visibles entre sí sin calificación. Los identificadores que están fuera del espacio de nombres pueden tener acceso a los miembros mediante el nombre completo de cada identificador, por ejemplo , o bien mediante una declaración using para un único identificador
11. Ejemplo:  
    #include <iostream>  
      
    namespace Mates {  
      
     int sumar(int op1, int op2)  
     {  
     return op1 + op2;  
     }  
    }  
      
    int main()

{

    std::cout << "¡Hola, mundo!" << std::endl;  
    std::cout << "3 y 2 son: " << Mates::sumar( 3, 2 ) << std::endl;

}

**Marco Practico:**

**Trabajo8.cpp**

#include <iostream>

using namespace std;

#include "lista.h"

#include "registro.h"

int main()

{

    lista::ListaDeAlumnos listaDeAlumnos;

    string nombre, apellido;

    int opcion;

    listaDeAlumnos.contador = 0;

    cargarLista(listaDeAlumnos);

    do

    {

        cout << "1 - Agregar un alumno" << endl;

        cout << "2 - Borrar a un alumno" << endl;

        cout << "3 - Mostrar la lista" << endl;

        cout << "4 - Ordenar la lista" << endl;

        cout << "0 - Salir" << endl;

        cin >> opcion;

        switch(opcion)

        {

            case 1:

                agregarAlumno(listaDeAlumnos);

                break;

            case 2:

                cout << "Ingrese el nombre del alumno que desee eliminar: ";

                cin >> nombre;

                cout << "Ingrese el apellido del alumno que desee eliminar: ";

                cin >> apellido;

                borrarAlumno(listaDeAlumnos, nombre, apellido);

                break;

            case 3:

                mostrarLista(listaDeAlumnos);

                break;

            case 4:

                cout << "5 - Por nombres descendente" << endl;

                cout << "6 - Por nombres ascendente" << endl;

                cin >> opcion;

                switch(opcion)

                    case 5:

                    ordenarDescendente(listaDeAlumnos);

                    break;

                    case 6:

                    ordenarAscendente(listaDeAlumnos);

                    break;

                break;

            case 0:

                guardarLista(listaDeAlumnos);

                break;

            default:

                cout << "Opcion no valida" << endl;

                break;

        }

    } while (opcion != 0);

    return 0;

}

Lista.cpp

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

#include "lista.h"

void lista::cargarLista(ListaDeAlumnos &listaDeAlumnos)

{

    ifstream file;

    file.open("Alumnos.txt", ios::in);

    if(!file.is\_open())

    {

        cout << "Error de apertura del archivo!" << endl;

    }

    else

    {

        while (!file.eof())

        {

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].nombre;

            file.get();

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].apellido;

            file.get();

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].carrera;

            file.get();

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].dni;

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].ano;

            if((!listaDeAlumnos.alumnos[listaDeAlumnos.contador].nombre.empty())&&(!listaDeAlumnos.alumnos[listaDeAlumnos.contador].apellido.empty())&&(!listaDeAlumnos.alumnos[listaDeAlumnos.contador].carrera.empty())&&(!listaDeAlumnos.alumnos[listaDeAlumnos.contador].dni.empty())&&(listaDeAlumnos.alumnos[listaDeAlumnos.contador].ano != 0))

            {

                listaDeAlumnos.contador++;

            }

        }

    }

    file.close();

}

void lista::guardarLista(ListaDeAlumnos &listaDeAlumnos)

{

    ofstream file;

    file.open("Alumnos.txt", ios::in);

    if(file.is\_open())

    {

        for(int i = 0; i < listaDeAlumnos.contador; i++)

        {

            file << listaDeAlumnos.alumnos[i].nombre<< " " << listaDeAlumnos.alumnos[i].apellido<< " " << listaDeAlumnos.alumnos[i].carrera<< " " << listaDeAlumnos.alumnos[i].dni<< " " << listaDeAlumnos.alumnos[i].ano<<endl;

        }

    }

    file.close();

}

void lista::agregarAlumno(ListaDeAlumnos &listaDeAlumnos)

{

    if(listaDeAlumnos.contador < MAX)

    {

        cout << "Ingrese el nombre del alumno: ";

        cin.ignore();

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].nombre);

        cout << "Ingrese el apellido del alumno: ";

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].apellido);

        cout << "Ingrese la carrera del alumno: ";

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].carrera);

        cout << "Ingrese el numero de DNI del alumno: ";

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].dni);

        cout << "Ingrese en que ano de la carrera esta el alumno: ";

        cin >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].ano;

        listaDeAlumnos.contador = (listaDeAlumnos.contador + 1);

    }

    else

    {

        cout << "Maximo de alumnos en la lista alcanzado (100!)" << endl;

    }

}

void lista::borrarAlumno(ListaDeAlumnos &listaDeAlumnos, string nombre, string apellido)

{

    int pos = 0;

    for(int i = 0; i < listaDeAlumnos.contador; i++)

    {

        if((listaDeAlumnos.alumnos[i].nombre == nombre)&&(listaDeAlumnos.alumnos[i].apellido == apellido))

        {

            pos = i;

            for(i = pos; i < listaDeAlumnos.contador -1; i++)

            {

                listaDeAlumnos.alumnos[i] = listaDeAlumnos.alumnos[i+1];

            }

            listaDeAlumnos.contador = (listaDeAlumnos.contador -1);

            cout << "Alumno eliminado correctamente!" << endl;

        }

    }

}

void lista::mostrarLista(ListaDeAlumnos &listaDeAlumnos)

{

    string nombre, apellido, carrera, dni;

    int ano;

    cout << "Listado de estudiantes: " << endl;

    for(int i = 0; i < listaDeAlumnos.contador; i++)

    {

        nombre = listaDeAlumnos.alumnos[i].nombre;

        apellido = listaDeAlumnos.alumnos[i].apellido;

        carrera = listaDeAlumnos.alumnos[i].carrera;

        dni = listaDeAlumnos.alumnos[i].dni;

        ano = listaDeAlumnos.alumnos[i].ano;

        cout << "Alumno: " << nombre << " " << apellido << ". DNI: " << dni << " De la carrera: " << carrera << " año: " << ano << endl;

        cout << endl;

    }

}

void lista::ordenarDescendente(ListaDeAlumnos &listaDeAlumnos)

{

    Alumno tmp;

    for(int i = 0; i < listaDeAlumnos.contador - 1; i++)

    {

        for(int j = listaDeAlumnos.contador - 1; j > i; j--)

        {

            if(listaDeAlumnos.alumnos[j] < listaDeAlumnos.alumnos[j - 1])

            {

                tmp = listaDeAlumnos.alumnos[j];

                listaDeAlumnos.alumnos[j] = listaDeAlumnos.alumnos[j - 1];

                listaDeAlumnos.alumnos[j - 1] = tmp;

            }

        }

    }

}

void lista::ordenarAscendente(ListaDeAlumnos &listaDeAlumnos)

{

    Alumno tmp;

    for(int i = 0; i < listaDeAlumnos.contador - 1; i++)

    {

        for(int j = listaDeAlumnos.contador - 1; j > i; j--)

        {

            if(listaDeAlumnos.alumnos[j] > listaDeAlumnos.alumnos[j - 1])

            {

                tmp = listaDeAlumnos.alumnos[j];

                listaDeAlumnos.alumnos[j] = listaDeAlumnos.alumnos[j - 1];

                listaDeAlumnos.alumnos[j - 1] = tmp;

            }

        }

    }

}

Lista.h

#ifndef lista\_h

#define lista\_h

#include <string>

using namespace std;

#include "registro.h"

namespace lista{

const int MAX = 100;

//Array del registro

typedef Alumno Alumnos[MAX];

//Lista y array juntos,con un contador

typedef struct

{

    Alumnos alumnos;

    int contador;

} ListaDeAlumnos;

void cargarLista(ListaDeAlumnos &listaDeAlumnos);//Funcion que carga toda la lista de la base al array

void guardarLista(ListaDeAlumnos &listaDeAlumnos);//Funcion que guarda en la base los cambios en el array

void agregarAlumno(ListaDeAlumnos &listaDeAlumnos);//Funcion para agregar alumnos

void borrarAlumno(ListaDeAlumnos &listaDeAlumnos, string nombre, string apellido);//Funcion para eliminar a un alumno

void mostrarLista(ListaDeAlumnos &listaDeAlumnos);//Muestra la liesta en pantalla

void ordenarDescendente(ListaDeAlumnos &listaDeAlumnos);//Ordena por nombre en orden descendente

void ordenarAscendente(ListaDeAlumnos &listaDeAlumnos);//Ordena por nombre en orden Ascendente

} //espacio de nombre lista

#endif //lista\_h

Registro.cpp

#include <iostream>

#include <string>

using namespace std;

#include "registro.h"

bool lista::operator<(Alumno opIzq, Alumno opDer)

{

    return (opIzq.nombre < opDer.nombre);

}

bool lista::operator>(Alumno opIzq, Alumno opDer)

{

    return (opIzq.nombre > opDer.nombre);

}

Registro.h

#ifndef registro\_h

#define registro\_h

#include <string>

using namespace std;

namespace lista{

//Estructura para los datos individuales de la lista

typedef struct

{

    string nombre, apellido, carrera, dni;

    int ano;

} Alumno;

bool operator>(Alumno opIzq, Alumno opDer);//Sobrecarga de operador para comparar por los nombres

bool operator<(Alumno opIzq, Alumno opDer);//Otra sorbecarga de operador para comparar por los nombres

} //espacio de nombre lista

#endif //registro\_h

