Normas Generales

- La práctica es **individual** y consiste en la implementación completa de la solución a un problema. Supondrá **2 puntos** sobre la nota final de la asignatura.
- La resolución se hará en ISO C++. Para la resolución, prioritariamente, se utilizarán los conocimientos adquiridos en toda la asignatura.
- En la evaluación se tendrá en cuenta, además de la corrección de la solución propuesta, el estilo de programación, el uso correcto de espacios y *tabuladores*, así como la claridad del código fuente.
- Debe incluir, una descripción de los principales procesos llevados a cabo dentro del algoritmo implementado (usando comentarios).
- Se aconseja incluir *filtros* para la lectura de las variables de entrada.
- Se habilitará una entrega en PRADO para poder subir el código fuente del programa. **La fecha límite es el 22 de enero de 2021 a las 23.55h.**

◁ Ejercicio 1 ▷ Clase Programa_LPS [2 puntos]

Se pretende trabajar con un lenguaje de programación simplificado, que vamos a denominar *LPS*. En el lenguaje *LPS*, sólo se pueden usar, como máximo, 1000 variables enteras (que se referencian con un número entre 1 y 1000), y los programas consistirán en una serie de instrucciones ordenadas con las siguientes características:

1. Considere que `v, v1, v2, v3` $\in \{1, \cdots, 1000\}$, `p` $\in N$ y `e` $\in Z$.
2. **(Inicializar)** `INI v e`: Asigna un valor entero e a la variable `v`.
3. **(Asignar)** `ASIG v1 v2`: Asigna el contenido de la variable `v2` a la variable `v1`.
4. **(Indirección)** `IND v1 v2`: Asigna a la variable `v1` el contenido de la variable cuya referencia se almacena en la variable `v2`.
5. **(Sumar)** `SUM v1 v2 v3`: Cambia el valor de la variable `v1` por la suma de los valores de las variables `v2` y `v3`.
6. **(Incrementar)** `INC v`: Incrementa el valor de la variable `v` en uno.
7. **(Multiplicar)** `MULT v1 v2 v3`: Cambia el valor de la variable `v1` por la multiplicación de los valores de las variables `v2` y `v3`.
8. **(Ir si menor)** `IR< v1 v2 p`: Esta instrucción permite cambiar el orden secuencial de ejecución de las instrucciones. En concreto, si se cumple que `v1` $<$ `v2`, entonces la siguiente instrucción a ejecutar será la que ocupa la posición `p`-ésima en el programa (las posiciones comienzan en 1). En otro caso, la ejecución continuará con la siguiente instrucción.
9. **(Ir si igual)** `IR= v1 v2 p`: Si las variables `v1` y `v2` contienen el mismo valor, entonces la siguiente instrucción a ejecutar será la que ocupa la posición `p`-ésima en el programa. En otro caso, la ejecución continuará con la siguiente instrucción.
10. **(Ir)** `IR p`: La siguiente instrucción a ejecutar será la que ocupa la posición `p`-ésima en el programa.
11. **(Finalizar)** `FIN`: Fuerza el final del programa.

Para materializar este lenguaje, se propone crear la clase `Programa_LPS`. Los métodos principales de esta clase serán:

```
int main(){
       int n;

       do{
          cout << "Introducir entrada: ";
          cin >> n;
       }while (n <= 0);

       Programa_LPS programa;
       programa.SetVar(1, n);

/*1*/  programa.Instruccion("INI", 3, 1, 0);
/*2*/  programa.Instruccion("INI", 2, 1, 0);
/*3*/  programa.Instruccion("MULT",3, 3, 2);
/*4*/  programa.Instruccion("INC", 2, 0, 0);
/*5*/  programa.Instruccion("IR<", 2, 1, 3);
/*6*/  programa.Instruccion("IR=", 2, 1, 3);
/*7*/  programa.Instruccion("FIN", 0, 0, 0);

       int error = programa.Ejecutar();
       if ( error == 0 )
          cout << "Salida del programa: " << programa.GetVar(3) << endl;
       else
          cout << "Programa detenido: existe error de sintaxis" << endl;
}
```

Figure 1: Ejemplo 1

1. `Constructor()`: Crea un programa vacío, con todas las variables a cero.
2. `void SetVar(int i, int e)`: Actualiza el valor de la variable `i` a `e`.
3. `void Instruccion (string inst, int e1, int e2, int e3)`: Introduce una nueva instrucción al final de un programa. Cuando una instrucción usa menos de 3 parámetros (como `INC`), se complementará la llamada con ceros. El máximo número de instrucciones será 100.
4. `int GetVar(int i)`: Devuelve el valor de la variable `i`.
5. `int Ejecutar()`: Ejecuta un programa, actualizando las variables asociadas. Devolverá 0 si el programa se ha ejecutado correctamente. Si existen errores de sintaxis (por ejemplo, la instrucción no existe o la referencia a una variable es negativa), el programa se detiene y devuelve 1.

Como primer ejemplo, mostramos un `main` en el que se procesa un programa en *LPS* que, dado un valor entero positivo n ($n \geq 1$), devuelve $n!$ (Figura 1). Como segundo ejemplo, presentamos un `main` en el que se incluye un programa en *LPS* que, dados 100 enteros, obtiene el máximo de ellos (Figura 2).

El alumno debe abordar las siguientes tareas:

1. Desarrollar la implementación de la clase `Programa_LPS`: atributos y métodos. (**0,75 puntos**)
2. Probar que los dos ejemplos funcionan correctamente. (**0,25 puntos**)
3. Implementar un programa principal que incluya el uso de la clase `Programa_LPS` para que dado un entero positivo n ($n \geq 1$), calcule $\sum_{i=1}^{n} i^2$. (**0,25 puntos**)
4. Construir un programa principal que gestione la clase `Programa_LPS` para determinar si un entero específico se encuentra entre 500 enteros (la respuesta será SI o NO). (**0,75 puntos**)

Así, se deben entregar, en total, cinco ficheros cpp (uno por cada tarea).

```
int main(){
       int x;

       Programa_LPS programa;

       // Leer 100 números desde consola y
       // suministrarlos a 'programa'.
       for (int i=1; i<=100; i++){
          cout << "Introducir entrada: " ;
          cin >> x;
          programa.SetVar(i, x);
       }

       // Suministrar instrucciones a 'programa'
/* 1*/ programa.Instruccion("INI",  101, 100, 0);
/* 2*/ programa.Instruccion("INI",  102, 1, 0);
/* 3*/ programa.Instruccion("ASIG", 103, 1, 0);
/* 4*/ programa.Instruccion("INC",  102, 0, 0);
/* 5*/ programa.Instruccion("IR<",  101, 102, 10);
/* 6*/ programa.Instruccion("IND",  104, 102, 0);
/* 7*/ programa.Instruccion("IR<",  104, 103, 4);
/* 8*/ programa.Instruccion("ASIG", 103, 104, 0);
/* 9*/ programa.Instruccion("IR",   4, 0, 0);
/*10*/ programa.Instruccion("FIN",  0, 0, 0);

       // Ejecutar 'programa' y dar salidas por consola
       int error = programa.Ejecutar();
       if ( error == 0 )
          cout << "Salida del programa: " << programa.GetVar(103) << endl;
       else
          cout << "Programa detenido: existe error de sintaxis" << endl;
}
```

Figure 2: Ejemplo 2