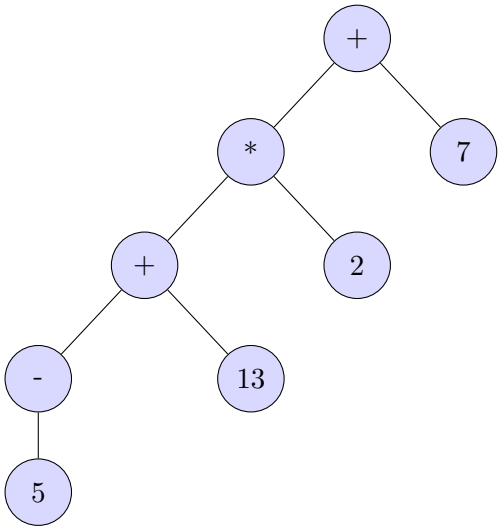
Trabajo práctico I

1 Motivación:

En un *árbol de expresiones aritméticas* las hojas corresponden a los operandos de la expresión (variables o constantes), mientras que los nodos restantes contienen operadores. Dado que los operadores matemáticos son binarios (como el caso de la suma) o unarios (como en el caso del operador negación), un árbol de expresiones resulta ser un árbol binario. Por ejemplo, el siguiente árbol de expresiones aritméticas



se corresponde con la expresión

$$(-5 + 13) * 2 + 7$$

y cuya evaluación da como resultado 23.

El objetivo del trabajo es implementar un intérprete que evalúe árboles de expresiones aritméticas, donde los operadores pueden ser binarios o unarios y los operandos, números enteros.

2 Entrada:

Las expresiones aritméticas que recibe el programa están dadas en *notación postfija*, es decir, los operadores se escriben después que sus operandos. Por ejemplo, la expresión anterior sería

$$5 - 13 + 2 * 7 +$$

La ventaja de esta notación es que no requiere de paréntesis ni de precedencias, dado que no hay ambigüedades en su procesamiento. Simplemente basta con saber la aridad de los operadores, esto es, si son binarios o unarios.
El único operador que requiere de un trato diferencial es el símbolo menos, dado que puede ser usado tanto de forma binaria como unaria. Para distinguirlos, usamos el símbolo -- para denotar el operador que calcula el opuesto de un número y reservamos el - para la resta.

3 Tabla de operadores

La tarea de procesar la entrada requiere conocer de antemano los operadores involucrados en la expresión. Para este fin, se hace uso de una tabla de operadores. Cada entrada de la tabla se corresponde con un operador y contiene la siguiente información:

- una cadena de caracteres con el símbolo del operador,
- un número con la aridad del operador, y
- un puntero a una función de evaluación.

Por ejemplo, la suma se representa por "+", su aridad es 2, y su función de evaluación toma dos enteros y retorna la suma de ellos.

4 Funcionamiento:

El funcionamiento del programa se puede dividir en dos partes, una estática y una interactiva. En la primera, se cargan manualmente los operadores en la tabla. Luego de esto, se ejecuta la parte interactiva, encargada de interpretar los comandos ingresados por el usuario por entrada estándar.

4.1 Carga de operadores:

El programa debe proveer una función

```
void cargar_operador(TablaOps *tabla, char *simbolo, int aridad, FuncionEvaluacion eval)
```

que permita cargar un nuevo operador a la tabla.

Las funciones de evaluación deben respetar la siguiente interfaz.

```
#ifndef __OPERADORES_H__
#define __OPERADORES_H__

typedef int (*FuncionEvaluacion)(int *args);

int suma(int *args);
int resta(int *args);
int opuesto(int *args);
int producto(int *args);
int division(int *args);
int modulo(int *args);
int potencia(int *args);

#endif /** __OPERADORES_H__ */
```

Es **obligatorio** utilizar los siguientes símbolos: "+" (suma), "-" (resta), "--" (opuesto), "*" (producto), "/" (división), "%" (módulo), y "^" (potencia).

4.2 Intérprete:

Una vez cargados todos los operadores, se llama a una función

```
void interpretar(TablaOps *tabla)
```

que da inicio a la parte interactiva del programa. En la misma, el programa lee por entrada estándar los comandos ingresados por el usuario y los resuelve utilizando los datos de la tabla de operadores. Los comandos soportados por el intérprete se detallan a continuación:

a) `ALIAS = cargar EXPR`

Procesa la string `EXPR` que contiene un expresión aritmética en notación postfija, aloca en memoria su árbol de expresiones aritméticas correspondiente y asocia esta dirección de memoria con la string `ALIAS`. La string `ALIAS` solo puede contener caracteres alfanuméricos, no puede comenzar con un número y no puede ser igual a ninguna de las palabras reservadas: cargar, imprimir, evaluar, y salir.

b) `imprimir ALIAS`

Imprime por pantalla la expresión aritmética en *notación infija* correspondiente al árbol asociado a la string `ALIAS`. La notación infija es la notación tradicional usada en matemática y pueden requerirse de paréntesis para indicar el orden de evaluación.

c) `evaluar ALIAS`

Evalúa el árbol asociado a la string `ALIAS` e imprime por pantalla el resultado.

d) `salir`

Finaliza el programa, liberando la memoria solicitada.

Ejemplo de ejecución:

```
> expr1 = cargar 5 -- 13 + 2 * 7 +
> imprimir expr1
(--5 + 13) * 2 + 7
> evaluar expr1
23
> salir
```

Nota: la salida de imprimir en general no es única, aunque la propuesta es la que usa la menor cantidad de paréntesis. En el trabajo puede implementar la salida que prefiera, procurando no cambiar la semántica de la expresión.

5 Consigna

a) Diseñe e implemente un programa que cumpla con las funcionalidades detalladas. Considere los siguientes puntos:

- El proptotipo de las funciones `cargar_operador` e `interpretar` es una sugerencia. Puede cambiarlo procurando mantener el nombre de la función.
- Los requerimientos no mencionados en el trabajo deben ser resueltos a criterio de cada grupo. Entre ellos: manejo de errores del intérprete (errores de sintaxis en los comandos, en las expresiones o en los alias, uso de operadores o de alias no definidos, división por cero, etc), redefinición de alias, paréntesis al imprimir.
- Respete las convenciones de código y proyecto elaboradas por la cátedra y disponibles en comunidades.

- Consultas: **únicamente** por mensaje privado de zulip. Usar como destinatario `edya1`.

b) Incluya en la entrega los casos de prueba usados para probar su programa.

c) Confeccione un informe detallando en un párrafo las dificultades encontradas y cómo fueron resultas, un ejemplo de cómo compilar el programa, y la bibliografía consultada.