Estructuras de Datos y de la Información

Práctica 2

Evaluar Expresiones de Números Racionales

OBJETIVO: El objetivo de la práctica es implementar un tipo genérico de datos utilizando la sentencia `template` del lenguaje C++.

FECHA: Está práctica se entregará los días 21, 22 y 23 de octubre.

ENUNCIADO: Realizar un programa C++ que solicite una expresión de números racionales en notación in-fija, la evalúe y devuelva el resultado.

La expresión se almacena en una cadena de caracteres y se asocia a un objeto `istrstream` que permitirá extraer los números racionales y los operadores de la expresión utilizando el operador de extracción de un flujo (>>). (Ver Nota de Implementación 2)

```
#include <strstream.h>

const int MAX_SIZE = 100;
char expr_str[MAX_SIZE];                // Cadena de caracteres
cin.getline(expr_str, MAX_SIZE);        // Lee una linea
istrstream expr(expr_str, MAX_SIZE);    // Objeto flujo de entrada

Racional r;
expr >> r;             // Extrae un número Racional
cout << r << endl;     // y lo envía a pantalla
```

Los operadores reconocidos son: suma , resta, producto y cociente. La siguiente tabla recoge la precedencia de los operadores.

Operador	Precedencia
+ -	1
* /	2

Además se podrán utilizar paréntesis para expresar sub-expresiones. Los paréntesis se consideran operadores a los que no se asocia precedencia.

Para evaluar la expresión se utilizan dos estructuras tipo pila: una pila de operadores y una pila de valores. (Ver Nota de Implementación 1)

```
Pila<char> Ope;
Pila<Racional> Val;
```

El proceso de evaluación sigue las siguientes reglas:

- Ambas colas están vacías al iniciar el proceso.
- Los números racionales y los operadores se extraen uno por uno del flujo.
- Los números racionales extraídos del flujo se guardan directamente en la pila de valores.
- Los paréntesis izquierdos se guardan directamente en la pila de operadores.
- Los paréntesis derechos fuerzan a sacar y evaluar los operadores de la pila hasta encontrar un paréntesis izquierdo o se vacía la pila. Se quita el paréntesis izquierdo de la pila de operadores.
- La acción de sacar y evaluar un operador se describe con las siguientes sentencias:

```
x = Val.Pop();              // Sacar segundo operando
y = Val.Pop();              // Sacar primer operando
op = Ope.Pop();             // Sacar operador
z = calcular(x, y , op);    // Realiza la operación op sobre x, y
Val.Push(z);                // Guardar el resultado
```

- Cuando se extrae un operador del flujo, si la pila de operadores está vacía o el operador guardado en el top de la pila tiene menor precedencia, el operador se guarda directamente en la pila.
- Si se extrae del flujo un operador con menor o igual precedencia que el operador guardado en el top de la pila, se fuerza a sacar y evaluar los operadores de la pila hasta encontrar un operador de menor precedencia o un paréntesis izquierdo o se vacía la pila.
- Cuando se terminan los operandos del flujo de entrada, se fuerza a sacar y evaluar todos los operadores de la pila.
- El proceso finaliza correctamente cuando se terminan los operandos del flujo de entrada y se vacía la pila de operadores. En este caso, la pila de valores contiene el resultado de la expresión.
- El proceso finaliza en error cuando no se encuentran valores al aplicar la acción de sacar y evaluar un operador. Esto indica que la expresión es incorrecta.

NOTAS DE IMPLEMENTACION:

1. Utilizar la plantilla `Pila` guardada en el fichero `pila.h`

```
template <class Data>
class Pila {
  private:
    struct Nodo {            // Declaración del tipo Nodo
      Data dat;
      Nodo *ptr;
    } *Top;

  public:
    Pila() {Top = 0;}        // Constructor
    ~Pila();                 // Destructor
    bool Empty() const {return(Top == 0);}
    bool Push(Data d);       // Guarda un dato en la pila
    Data Pop();              // Saca un dato de la pila
    Data get() {if(¡Empty()) return Top->dat;} };

template<class Data> Pila<Data>::~Pila() {
  while(¡Empty()) Pop();
}

template<class Data> bool Pila<Data>::Push(Data d) {
  Nodo *ptr = new Nodo;
  if(ptr != 0) {
    ptr->dat = d;
    ptr->ptr = Top;
    Top = ptr;
    return 1;
  }
  return 0;
}

template<class Data> Data Pila<Data>::Pop() {
  if(!Empty()) {
    Nodo *ptr = Top;
    Data d = Top->dat;
    Top = Top->ptr;
    detete ptr;
    return d;
  }
  return 0;
}
```

2. Utilizar los métodos de la clase istream para inspeccionar el siguiente carácter del flujo.

```
istream& istream::putback(char ch); // Devuelve el char ch al flujo
istream& istream::get(char &ch);    //Extrae un carácter del flujo
istream& istream::getline(char* str, int len, char delim='\n');
```