Expresiones en notación polaca inversa

La notación polaca inversa (RPN) es una notación matemática para expresiones que consiste en que todos los operandos preceden a los operadores. Tiene varias ventajas para la evaluación de expresiones, como: los cálculos se realizan ni bien se especifica el operador, no necesitan la expresión entera para la evaluación, facilidad de evaluar expresiones de complejidad arbitraria y no se utilizan paréntesis.

Por ejemplo la expresión:

$$\frac{\sin(3x+4)}{2+3\cos\left(\frac{x}{2+x}\right)}$$

Escrita en RPN quedaría:

```
3 X * 4 + sin 2 3 X 2 X + / cos * + /
```

Dadas las clases para la representación de expresiones del ejercicio 5 de la práctica 14, se solicita:

a- Implementar una función que imprima la expresión en formato RPN.

```
void PrintExpRPN(Exp *expr);
```

b- Se desea representar directamente la expresión en RPN, para lo cual se define una clase base para ser especializada para distintos tipos de nodos en la representación:

```
class ExpRPN {
    public:
        virtual ~ExpRPN() {}
        virtual void print() = 0;
     protected:
        ExpRPN *pNext;        // siguiente nodo, nullptr si es el último
};
```

Con el mismo criterio de especialización seguido para los distintos tipos de **Exp** en el ejercicio 5 de la práctica 14 defina los tipos derivados de nodos para una expresión **ExpRPN**, e implementar una función que convierta una expresión del tipo **Exp** en una expresión del tipo **ExpRPN**. El prototipo de la función será:

```
ExpRPN *Exp2Rpn(Exp *expression);
```

Detalle de cuotas en un país sin inflación

Una forma común de realizar gastos es financiándolos en cuotas.

Se desea realizar un sistema de consultas que, a partir de una lista de gastos realizados a lo largo del tiempo (granularidad de meses) (cada gasto incluye el monto total y la cantidad de cuotas en las que se hizo la operación), se pueda obtener un detalle de las cuotas a vencer en un mes determinado.

Para ello se cuenta con un archivo de gastos “gastos.txt” que tiene los gastos realizados ordenados cronológicamente, el mismo tiene 4 columnas:

i. La primera indica el mes del gasto. Los meses toman números enteros, partiendo del mes 0. No existe división por año, de tal modo que existen los meses 13, 14, etc…

ii. La segunda columna indica el monto del gasto.

iii. La tercera columna indica si el gasto es de contado (valor 0) o si es en cuotas (valor >0). En el caso de ser en una cuota el gasto se acredita al mes siguiente. En general si el gasto es en N cuotas (N>=1), el gasto se acredita los siguientes N meses en valores del gasto dividido en N.

iv. La última columna contiene una única palabra que describe el gasto realizado.

Se solicita:

a. Leer el archivo gastos.txt e introducir sus valores en una variable del tipo:

```
struct GastoEntrada {
    int    mes;
    double valor;
    int    cuotas;
    string descripcion;
};

list<GastoEntrada> gastos;
```

b. Dado un mes (por ejemplo el mes 13), imprimir las cuotas a pagar dicho mes (mes, valor y descripción) y el monto total a pagar. Para esto filtre la lista de gastos para incluir todas las cuotas a pagar dicho mes. Por ejemplo, si en el mes 2 se realizó un pago en 12 cuotas de $1200 en concepto de facturas, en el mes 13 se deberá incluir un pago de $100 en concepto de facturas.

Aritmética entera de largo ilimitado

La aritmética entera de largo ilimitado consiste en un conjunto de algoritmos, funciones y estructuras de datos diseñados específicamente para manejar números que pueden tener una cantidad de dígitos arbitrarios (posiblemente muy grande).
Suponga que un número entero positivo de precisión arbitrario se representa a través de una lista como la siguiente:

```
using digit = char;

using ArbNumber = list<digit>;
```

En donde el nodo a la cabeza de la lista contiene el digito que corresponde a 10^0, el segundo nodo contiene el dígito de 10^1, el tercero el de 10^2 y así sucesivamente.

a. Implemente una función que calcule y retorne la suma de 2 números de precisión arbitraria. El prototipo de la función deberá ser:

```
ArbNumber AddArbNumber(const ArbNumber &N1, const ArbNumber  &N2);
```

b. Implemente una función que imprima un número de precisión infinita. El número debe ser impreso en orden natural, es decir los dígitos de mayor orden primero. Prototipo:

```
void PrintArbNumber(const ArbNumber &N1);
```

Ley de Zipf

La llamada ley de Zipf, formulada en la década de 1940 por George Kingsley Zipf, lingüista de la Universidad de Harvard, es una ley empírica según la cual en una determinada lengua la frecuencia de aparición de distintas palabras sigue una distribución que puede aproximarse por :

$$P_n = \frac{1}{n^a}$$

En donde P_n representa la frecuencia de la n-ésima palabra mas frecuente y el exponente "a" es un número real positivo, en general ligeramente superior a 1.

Esto significa que el segundo elemento se repetirá aproximadamente con una frecuencia de 1/2 de la del primero, el tercer elemento con una frecuencia de 1/3 y así sucesivamente.

Para comprobar esta ley se solicita:

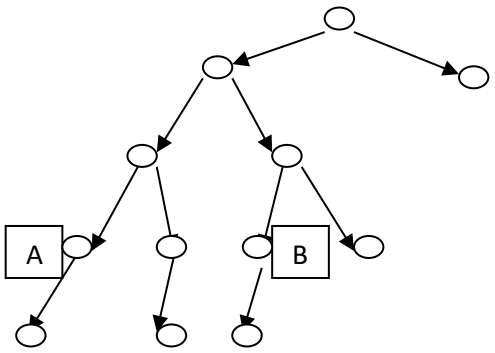
a. Utilizar un **`mapa<string,unsigned int>`** para ir colectando palabras (las mismas van a ser leídas desde un archivo). La clave de cada clave en el mapa será la palabra y el valor será el número de veces que la misma se repite.
b. Como haría para ordenar esos pares del mapa por el número de repeticiones de cada entrada?
c. Imprimir las frecuencias encontradas y a que palabras corresponden.

Distancia entre nodos de un árbol

Dado un diccionario binario del ejercicio 1 de la práctica 16, suponiendo que la distancia entre un nodo y un nodo hijo directo de este es 1, realizar una función y las auxiliares necesarias, que permitan calcular la distancia entre 2 nodos. Prototipo de la función requerida:

```
int DistanciaArbol(TreeNode *tree, int A, int B);
```

En el ejemplo, la distancia entre A y B es 4



Raíces enésimas

Se llama raíz enésima de la unidad a cualquiera de los números complejos que satisfacen la ecuación

$z^n = 1$ (n = 2, 3, 4, 5, 6, ...)

Las n diferentes raíces n-ésimas de la unidad, son los números

$e^{2\pi i\, k/n}$=cos($2\pi k/n$)+i sen($2\pi k/n$) (k = 0, 1, 2, …, n - 1).

La expresión de la izquierda es la fórmula de Euler, la de la derecha es la forma cartesiana.

Entre las raíces enésimas de la unidad siempre está el número 1, el número -1 solo está cuando n es par y los números i y -i cuando n es múltiplo de cuatro.

Dado un valor de n, se pide hallar todas las raíces enésimas de la unidad, utilizando la fórmula de Euler y escribiéndolas en forma cartesiana, utilizando para eso una estructura

```
class Complejo {
```

```
    public:
        Complejo(double r, double i);

        Complejo operator+(const Complejo &);

    private:
        double re;
        double im;
};
```

Comprobar numéricamente que la suma de todas las raíces enésimas de la unidad es igual a 0