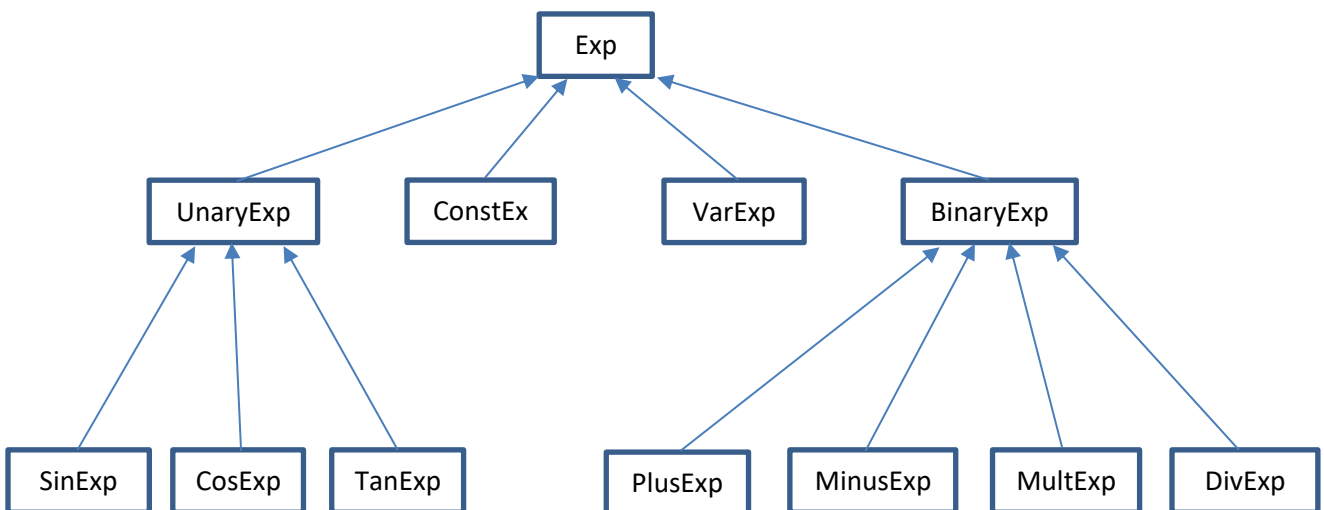
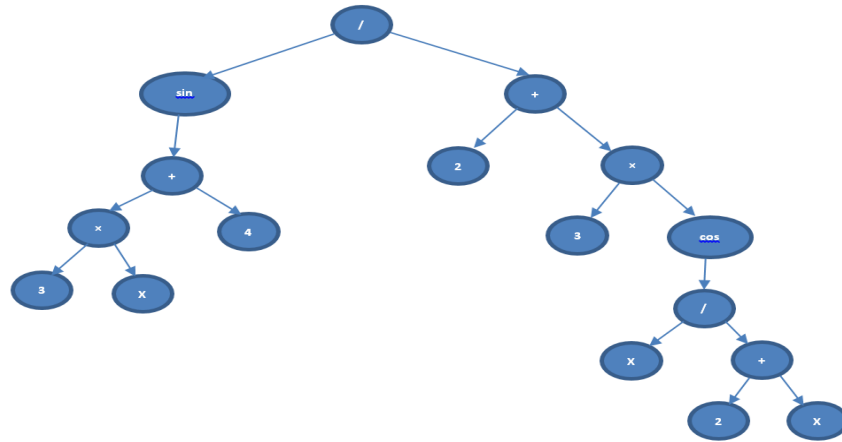


1. Cree una jerarquía de “figuras”: una clase base llamada **Figura** y clases derivadas, **Cuadrado**, **Circulo** y **Triangulo**. En la clase base haga una función virtual **dibujar()**, y sobrescríbala en las clases derivadas. Haga un arreglo nativo de punteros a **Figura** e inicialícelo con objetos creados dinámicamente en el heap (llamando al operador **new**), llame a **dibujar()** a través de punteros a la clase base para verificar el comportamiento de las funciones virtuales.
2. Modifique el ejercicio anterior para utilizar un **vector<Figura *>** en lugar de un arreglo nativo. Asegúrese que la memoria es liberada correctamente.
3. Modifique el ejercicio anterior definiendo el destructor como **virtual** y verifique que ocurre.
4. Cree una clase **Base** que tenga un dato miembro, y una clase derivada **Deriv** de ésta que agrega otro dato miembro. Escriba una función no miembro, **void f(Base b)**, que tome un objeto de la clase base por valor e imprima el tamaño del objeto utilizando **sizeof()**. En **main()** cree un objeto de la clase derivada, imprima su tamaño y llame a función **f**. Explique qué ocurre.
5. En el esqueleto de código presente en **expression.cpp** se implementa una jerarquía como la que se muestra en el diagrama siguiente para representar expresiones algebraicas en forma analítica:



Implemente los métodos faltantes y todo otro método que crea necesario. Pruebe el código con el **main** dado. En ese **main**, se está creando la expresión: $\frac{\sin(3x+4)}{2+3 \cos(\frac{x}{2+x})}$ que puede verse gráficamente como:



6. Una pila (stack en inglés) es una estructura de datos que permite almacenar y recuperar datos, siendo la semántica de acceso a sus elementos del tipo **LIFO** (del inglés **Last In, First Out**, «último en entrar, primero en salir»). Esta estructura es de uso frecuente en el área de informática debido a su simplicidad y capacidad de dar respuesta a numerosos problemas.

Para el manejo de los datos cuenta con dos operaciones básicas: **push**, que coloca un nuevo objeto en la pila, y su operación inversa, **pop**, que retira el último elemento apilado. Puede existir además una operación para chequear si el stack está o no vacío.

En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado **TOS: Top of Stack** en inglés). La operación retirar (**pop**) permite obtener este elemento, que es retirado de la pila permitiendo el acceso al anterior (apilado con anterioridad), que pasa a ser el último, el nuevo TOS.

Se solicita implementar el UDT **Stack** para almacenar caracteres con su interface básica:

```
class Stack {
    public:
        void push(char c); // coloca un nuevo carácter en la pila
        char pop();        // retira el elemento al tope del stack
        bool isEmpty();    // retorna true/false indicando si el stack está vacío
    private:
        // ...
        // ...
};
```