PROBLEMA 1

Especificar el TAD proposición con las siguientes operaciones:

- Crear una proposición con un único símbolo del conjunto A = {s1, s2, ..., sn}, con n constante.

- Crear una nueva proposición a partir de otra previamente creada y una conectiva ¬ (NO).

- Crear una nueva proposición a partir de otras previamente creadas y una conectiva que puede ser ∧ (Y) o ∨ (O).

- Calcular el valor de verdad de una proposición, a partir de una valoración del conjunto de símbolos A = {s1, s2, ..., sn}.

RESOLUCIÓN

TAD Proposicion

Operaciones:

`Proposicion(Simbolo s)`

- Poscondición: Crea una nueva proposición a partir del símbolo s

`Proposicion(const Proposicion& p, Conectiva op)`

- Precondición: op debe ser la negación
- Poscondición: Se crea una nueva proposición negando p

`Proposicion(const Proposicion& p1, const Proposicion& p2, Conectiva op)`

- Precondición: op debe ser el operador de conjunción o disyunción
- Poscondición: Se crea una nueva proposición a partir de p1 y p2 relacionados con el operador op

`bool ValordeVerdad(const Valoracion& vs)`

- Precondición: vs debe tener tantos elementos como símbolos haya en el conjunto A
- Poscondición: Devuelve la valoración de la proposición a partir de los valores de verdad de los símbolos que se encuentren en la proposición

PROBLEMA 2

Definir la representación de los siguientes tipos de datos:

- Símbolo
- Conectiva
- Valoración de un conjunto de n símbolos.
- Proposición

RESOLUCIÓN

```
typedef int Simbolo; //Solo nos interesa el subíndice del símbolo para su representación
enum Conectiva{NO, Y, O}; //Solo queremos esas tres

typedef vector<bool> Valoracion; //La valoración será un vector de true o false de tal forma
que Valoracion[0] será la valoración de s1, Valoracion[1] la de s2...
```

```
class Elemento_logico
{
    public:
        Elemento_logico(){};
        Elemento_logico(Simbolo x): e{'s'}, i{x}{};
        Elemento_logico(Conectiva c)
        {
            if(c==0)//negación
                e = 'N';
            else
                if(c==1)//conjunción
                    e= 'Y';
                else //disyunción
                    e= 'O';
            i=-1;
        }
        const char caracter() const {return(e);}
        const int num() const {return(i);}

    private:
        char e; //Para el carácter
        int i;  //Para el Número
};

class Proposicion
{
    public:
        Proposicion(Simbolo s); //Creo una proposición a partir de un símbolo
        Proposicion(Proposicion& p, Conectiva); //Creo una proposición a partir de otra propo
sición y un símbolo que debe ser negación (NO)
        Proposicion(Proposicion& P1, Proposicion& P2, Conectiva op); //Creo una proposición a
 partir de otra proposición y un símbolo que debe ser Y o o
        bool ValordeVerdad(const Valoracion& vs); //Devuelve la valoración de la proposición
    private:
        Elemento_logico E_;
        Proposicion * izq_;
        Proposicion * drcho_;
        int n; //Número de símbolos del conjunto A
};
```

PROBLEMA 3

Implementar las operaciones del TAD proposición

RESOLUCIÓN

```
Proposicion::Proposicion(Simbolo s)
{
    Elemento_logico l(s);
    E_=l;
    drcho_= nullptr;
    izq_=nullptr;
};

Proposicion::Proposicion(Proposicion& p, Conectiva op)
{
    assert(op==Conectiva::NO);
    Elemento_logico n(op);

    E_=n;
    izq_= &p; //Lo meto en el izq, es irrelevante
    drcho_= nullptr;
}

Proposicion::Proposicion(Proposicion& p1, Proposicion& p2, Conectiva op)
{
    assert(op==Conectiva::Y || op==Conectiva::O);
    Elemento_logico l{op};
    E_=l;
    izq_=&p1;
    drcho_=&p2;  //Es irrelevante si lo pongo en el lado derecho o el izq
}

bool Proposicion::ValordeVerdad(const Valoracion& vs)
//Como para la recursividada no necesito ningún parámetro extra, puedo utilizar esta misma fu
nción
{
    if(E_.num()==-1) //El elemento es un operador lógico
    {
        if(E_.caracter()== 'Y')
            return(izq_->ValordeVerdad(vs) && drcho_->ValordeVerdad(vs));
        else
            if(E_.caracter()== 'O')
                return(izq_->ValordeVerdad(vs) || drcho_->ValordeVerdad(vs));
            else //negación
                return(!izq_->ValordeVerdad(vs));
          //Cuando negamos lo hemos metido en el izq el resto del árbol
    }
    else //El elemento es un símbolo
        return(vs[E_.num()-1]);
```

```
//El valor de verdad de s1 estará en la posición 0 del vector, S2 en la 1, y así.
}
```

Os dejo por aquí la resolución de este examen que fue muy interesante. En lugar de tener que hacer una funcionalidad para los árboles con los que siempre trabajamos, tuvimos que crear nuestro propio TAD que en sí utilizaba una estructura de árbol. Si os fijáis he hecho lo mismo que en la implementación de los ABBs de clase. El subárbol tanto izquierdo como derecho están representados como tal como punteros a estructuras Proposición. Fijaos en que me cree una estructura auxiliar, Elemento Lógico para poder encapsular en un mismo tipo de dato, las conectivas y los símbolos, lo que me facilitó muchísimo la valoración de las proposiciones y la construcción de las mismas.

Para árboles la clave es pensar de forma recursiva. Al fin y al cabo un árbol está compuesto de más árboles y así sucesivamente. Todo lo relacionado con árboles es infinitamente más sencillo de forma recursiva.

Nada más que añadir. Mucha suerte y a pensar en recursivo.

¡Mucho ánimo!