Laboratorio 3

Estructuras de Datos Avanzadas
Universidad Nacional de San Agustín
Nombre: Alexander Rusvell Apaza Torres

1. Transformar expresión postfija a un árbol de expresión
   a) Código fuente:
      i. Nodo padre: De este nodo heredarán 2 hijos: iNodo y sNodo. De esta manera podremos obtener el resultado de la expresión con la función *getValue().*

```
class Nodo
{
     protected:
          Nodo* izq;
          Nodo* der;
          int id;
     public:
          Nodo();
          virtual int getValue() = 0;
          virtual void getRawValue() = 0;
          virtual string getStrValue() = 0;
          ~Nodo();
};

Nodo::Nodo()
{
     this->der = 0;
     this->izq = 0;
}
```

      ii. iNodo: Este nodo almacenará los números de la expresión. Hereda de Nodo.

```
class iNodo : public Nodo
{
     private:
          int value;
     public:
          iNodo(int value);
          int getValue() override;
          void getRawValue() override;
          string getStrValue() override;
          ~iNodo();
     friend class Arbol;
};

iNodo::iNodo(int value)
{
     this->value = value;
}
int iNodo::getValue(){
     return this->value;
}
void iNodo::getRawValue(){
     cout<<this->value;
}
string iNodo::getStrValue(){
     return to_string(this->value);
}
```

iii. sNodo: Este nodo almacenará los operadores de la expresión. También hereda de Nodo.

```
class sNodo : public Nodo {
     private:
          string value;
     public:
          sNodo(string value);
          int getValue() override;
          void getRawValue() override;
          string getStrValue() override;
          ~sNodo();
     friend class Arbol;
};
sNodo::sNodo(string value){
     this->value = value;
}
int sNodo::getValue(){
     if (this->value == "+")
          return this->izq->getValue() + this->der->getValue();
     else if (this->value == "-")
          return this->izq->getValue() - this->der->getValue();
     else if (this->value == "*")
          return this->izq->getValue() * this->der->getValue();
}
string sNodo::getStrValue(){
     return this->value;
}
```

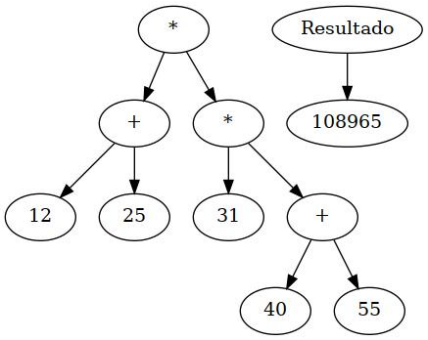
iv. Árbol: Aquí se genera el arbol de expresión. Recibe una lista con los elementos de la expresión en notación postfija y la transforma en un árbol de expresión. También generá un .dot para observar el árbol en sí y su resultado.

```
class Arbol {
     private:
          Nodo* root;
          list<Nodo*> pila;
     public:
          Arbol(list<string>);
          void generateDot(string&, Nodo*);
          void setIDs(Nodo* t, int&, string&);
          string getDot();
          ~Arbol();
};
Arbol::Arbol(list<string> post){
     int v;
     string vs;
     while (!post.empty()){
          if(isdigit(post.front()[0])){
               v = stoi(post.front());
               pila.push_back(new iNodo(v));
               post.pop_front();
          }
          else{
               vs = post.front();
               Nodo* nuevo = new sNodo(vs);
               nuevo->der = this->pila.back();
               this->pila.pop_back();
               nuevo->izq = this->pila.back();
               this->pila.pop_back();
               pila.push_back(nuevo);
               post.pop_front();
          }
     }
     if (pila.size()==1){
          this->root = pila.front();
          this->pila.clear();
     }

}
```

```
void Arbol::generateDot(string& res, Nodo* t) {
     if (!t) return;

     if (t->izq) res = res + t->id + " -> " + t->izq->id + "; \n";
     if (t->der) res = res + t->id + " -> " + t->der->id + "; \n";
     generateDot(res, t->izq);
     generateDot(res, t->der);
}

void Arbol::setIDs(Nodo* t, int& i,string& res) {
     if (!t) return;
     t->id = i;
     res = res + i + " [ label = \"" + t->getStrValue() + "\" ]; \n";
     i = i + 1;
     setIDs(t->izq,i, res);
     setIDs(t->der,i, res);
}

string Arbol::getDot() {
     string res = "digraph G {\n";
     int i = 0;
     this->setIDs(this->root, i, res);
     this->generateDot(res, this->root);
     res = res + "Resultado -> " + this->root->getValue() + "; \n";
     res = res + "}";
     return res;
}
```

v. Resultado: El programa en sí imprime el .dot, pero se captura la salida en un archivo de texto y luego se ejecuta el comando para generar la imagen

```
int main(int argc, char const *argv[]) {
     list<string> post = {"12","25","+","31","40","55","+","*","*"};
     Arbol c(post);
     cout<<c.getDot()<<endl;
     return 0;
}
```



b) Mostrar un árbol binario usando graphViz:

i. Código:

```
template<typename T>
class Arbol
{
     private:
          Nodo<T>* root;
     public:
          Arbol(){
               this->root = 0;
          }

          void push(T value, Nodo<T>*& r){
               if (!r){
                    r = new Nodo<T>(value);
                    return;
               }
               else{
                    push(value,r->children[value > r->value ]);
               }
          }
```

```
    void push(T value){
        this->push(value, this->root);
    }

    void generateDot(string& res, Nodo<T>* t){
        if (!t) return;

        if (t->children[0]) {
            res = res + t->value + " -> " + t->children[0]->value + "; \n";
        }
        if (t->children[1]) {
            res = res + t->value + " -> " + t->children[1]->value + "; \n";
        }
        generateDot(res, t->children[0]);
        generateDot(res, t->children[1]);
    }

    string getDot(){
        string res = "digraph G {\n";
        this->generateDot(res, this->root);
        res = res + "}";
        return res;
    }

};
```

ii. Resultado: Se hizo lo mismo que en el ejercicio anterior. Se imprimió y se capturo el resultado en un archivo de texto. De ahí se uso el comando para generar la imagen a partir del archivo .dot.

```
Arbol<int> a;
vector<int> v = {11,6,8,19,4,10,5,50,17,43,49,31};
for (auto i : v)
    a.push(i);
cout<<a.getDot()<<endl;
return 0;
```