```
#include <iostream>
#include <vector>
using namespace std;

class CompositeIterator;
class Composite;

class Component
{
public:
    virtual void traverse(){};
    virtual ComponentIterator* CreateIterator(){};
};

class Leaf : public Component
{
private:
    string value;
public:
    Leaf(string val)
    {
        value = val;
    }
    void traverse()
    {
        cout << value << ' ';
    }
};

class Composite: public Component
{

    vector < Component * > leaves;
public:

    void add(Component *node)
    {
        leaves.push_back(node);
    }
    int getCount () const {return leaves.size(); }
    Leaf * get(int index) const { return leaves[index];};
    void traverse()
    {
        for (int i = 0; i < leaves.size(); i++)
            leaves[i]->traverse();
    }
    CompositeIterator* CreateIterator() {
        return new CompositeIterator(this);
};

class ComponentIterator {
public:
    virtual void First() = 0;
    virtual void Next() = 0;
    virtual bool IsDone () const = 0;
    virtual Leaf* CurrentItem() const = 0 ;
protected:
    ComponentIterator(){};
};
```

```
    class ComponentIterator;

    class CompositeIterator : public ComponentIterator {
public:
    CompositeIterator(const Composite *composite)
 : _composite(composite), _current(0) {}
    void First(){_current = 0;};
    void Next(){_current++;};
    Leaf* CurrentItem() const{
           return (IsDone()?NULL:_composite->get(_current));
      };
    bool IsDone()const ;
        bool IsDone() const {
            return _current >= _composite->getCount();
        }
private:
    const Composite *_composite;
    int _current;
};

    void printAggregate(ComponentIterator& i) {
        cout << "Iterating over collection:" << endl;
        for(i.First();  !i.IsDone(); i.Next()) {
            cout << i.CurrentItem()->traverse() << endl;
        }
        cout << endl;
    }

int main()
{
    Component *component;
    component = new Composite();

    Leaf *root = new Leaf("expression");
    Leaf *expr_left = new Leaf("expression");
    Leaf *term_left = new Leaf("term");
    Leaf *const_left = new Leaf("5");
    Leaf *term = new Leaf("-");
    Leaf *right_right_term = new Leaf("term");
    Leaf *right_left_const = new Leaf("4");
    Leaf *right_term= new Leaf("/");erat
    Leaf *right_right_const = new Leaf("2");

    ComponentIterator *iterator = component->CreateIterator();


    component.add(const_left);
    component.add(term);
    component.add(right_left_const);
    component.add(right_term);
    component.add(right_right_const);

    printAggregate(*iterator);
    delete iterator;



}
```