Practical 1

18BCE243

Code of Practical 1

```
#include <set>
#include <stdexcept>

class Graph {
public:
    std::set<int> V;
    std::set<std::pair<int, int> > E;

    Graph() {}
    Graph(std::set<int> _V, std::set<std::pair<int, int> > _E)
    {
        for ( auto it = _E.begin() ; it != _E.end() ; ++it ) {
            auto v1 = it->first;
            auto v2 = it->second;

            if ( _V.find(v1) == _V.end() || _V.find(v2) == _V.end() ) {
                throw std::logic_error("error: The edge set contains"\
                                       "vertices not present in the "\
                                       "vertex set!");
            }
        }

        V = _V;
        E = _E;
    }

    Graph graph_union(const Graph &G2) {
        auto V1 = this->V;
        auto V2 = G2.V;
        auto E1 = this->E;
        auto E2 = G2.E;

        auto V3 = __get_union(V1, V2);

        auto E3 = __get_union(E1, E2);

        return Graph(V3, E3);
    }

    Graph graph_intersection(const Graph &G2) {
        auto V1 = this->V;
        auto V2 = G2.V;
        auto E1 = this->E;
        auto E2 = G2.E;

        auto V3 = __get_intersection(V1, V2);

        auto E3 = __get_intersection(E1, E2);

        return Graph(V3, E3);
    }

    Graph graph_ringsum(const Graph &G2) {
        auto V1 = this->V;
        auto V2 = G2.V;
        auto E1 = this->E;
        auto E2 = G2.E;

        auto V3 = __get_union(V1, V2);

        auto E3tmp1 = __get_union(E1, E2);
        auto E3tmp2 = __get_intersection(E1, E2);
        auto E3     = __get_difference(E3tmp1, E3tmp2);

        return Graph(V3, E3);
    }

    friend std::ostream& operator<<(std::ostream &fout, const Graph &_G)
    {
        fout << "Vertex Set: { ";
        for ( auto it = _G.V.begin() ; it != _G.V.end() ; ++it ) {
            fout << *it << " ";
        }
        fout << "}\n";
        fout << "Edge Set: { ";
        for ( auto it = _G.E.begin() ; it != _G.E.end() ; ++it ) {
            fout << "{" << it->first << ", " << it->second << "} ";
        }
        fout << "}";

        return fout;
    }

private:
    template <typename T>
    static std::set<T> __get_union(const std::set<T> &a, const std::set<T> &b)
    {
        std::set<T> result = a;
        result.insert(b.begin(), b.end());
        return result;
    }

    template <typename T>
    static std::set<T> __get_intersection(const std::set<T>& a, const std::set<T>& b)
    {
        std::set<T> tmp = __get_union(a, b);
        std::set<T> result;
        for ( auto it = tmp.begin() ; it != tmp.end() ; ++it )
        {
            if ( a.find(*it) != a.end() && b.find(*it) != b.end() ) {
                result.insert(*it);
            }
        }
        return result;
    }

    template <typename T>
    static std::set<T> __get_difference(const std::set<T>& a, const std::set<T>& b)
    {
        std::set<T> result;
        for ( auto it = a.begin() ; it != a.end() ; ++it )
        {
            if ( b.find(*it) == b.end() ) {
                result.insert(*it);
            }
        }
        return result;
    }
};
```

Test Driver (with Inputs)

```
#include <iostream>
#include "practical1.h"

int main()
{
    std::set<int> V1 = {1, 2, 3, 4};
    std::set<std::pair<int, int> > E1 = {{1, 2}, {2, 3}, {3, 4}, {4, 1}};
    auto G1 = Graph(V1, E1);
    std::set<int> V2 = {1, 2, 3, 4};
    std::set<std::pair<int, int> > E2 = {{1, 3}, {2, 4}};
    auto G2 = Graph(V2, E2);

    auto G3 = G1.graph_union(G2);
    auto G4 = G1.graph_intersection(G2);
    auto G5 = G1.graph_ringsum(G2);

    std::cout << "G1:\n";
    std::cout << G1 << std::endl;
    std::cout << "G2:\n";
    std::cout << G2 << std::endl;

    std::cout << "\nUnion of the G1 and G2 is : " << std::endl;
    std::cout << "G3:\n";
    std::cout << G3 << std::endl;

    std::cout << "\nIntersection of the G1 and G2 is : " << std::endl;
    std::cout << "G4:\n";
    std::cout << G4 << std::endl;

    std::cout << "\nRing Sum of the G1 and G2 is : " << std::endl;
    std::cout << "G5:\n";
    std::cout << G5 << std::endl;

    return 0;
}
```

Output

```
G1:
Vertex Set: { 1 2 3 4 }
Edge Set: { {1, 2} {2, 3} {3, 4} {4, 1} }
G2:
Vertex Set: { 1 2 3 4 }
Edge Set: { {1, 3} {2, 4} }

Union of the G1 and G2 is :
G3:
Vertex Set: { 1 2 3 4 }
Edge Set: { {1, 2} {1, 3} {2, 3} {2, 4} {3, 4} {4, 1} }

Intersection of the G1 and G2 is :
G4:
Vertex Set: { 1 2 3 4 }
Edge Set: { }

Ring Sum of the G1 and G2 is :
```

```
G5:
Vertex Set: { 1 2 3 4 }
Edge Set: { {1, 2} {1, 3} {2, 3} {2, 4} {3, 4} {4, 1} }
```