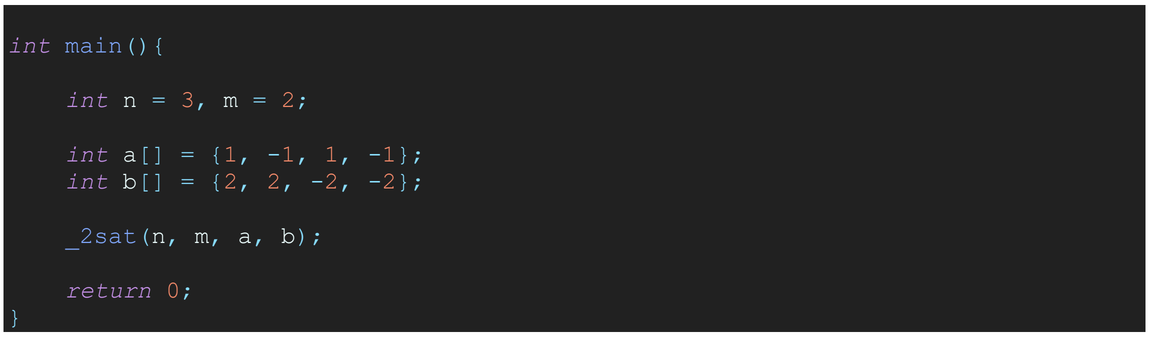
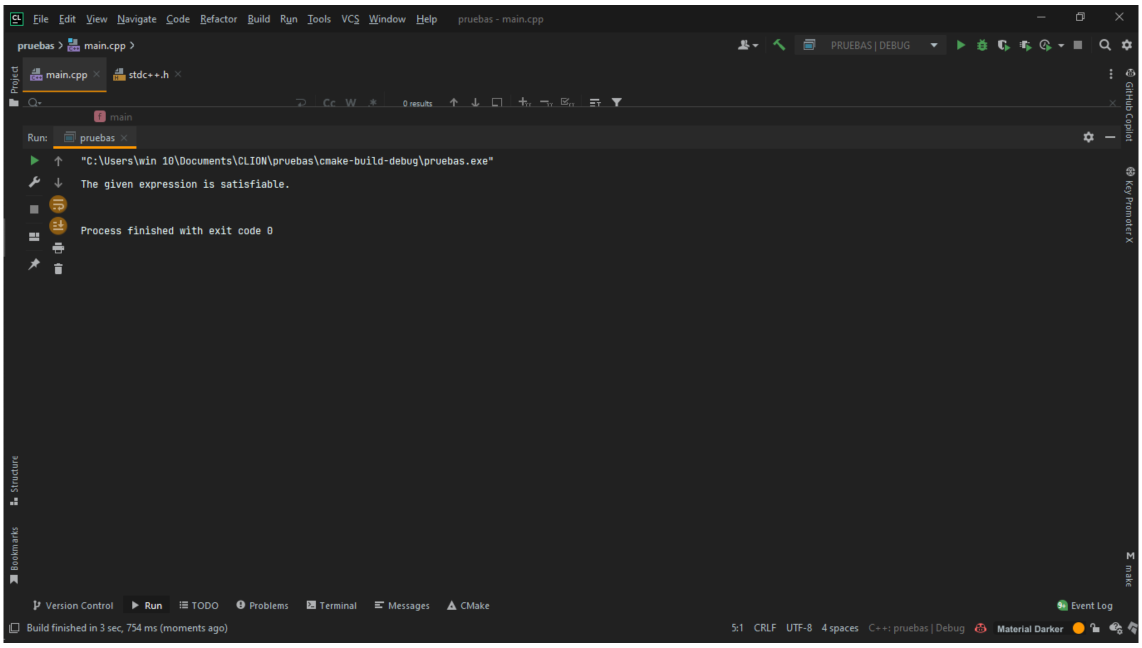
```cpp
#include <random>
#include <regex>
#include <iostream>

using namespace std;

const int MAX = 100000;

vector<int> adj[MAX];
vector<int> adjInv[MAX];
bool colored[MAX];
bool coloredInv[MAX];
stack<int> s;

int scc[MAX];

int counter = 1;

void addEdges(int a, int b)
{
    adj[a].push_back(b);
}

void addEdgesInverse(int a, int b)
{
    adjInv[b].push_back(a);
}

void dfs1(int u)
{
    if(colored[u])
        return;

    colored[u] = 1;

    for (int i=0;i<adj[u].size();i++)
        dfs1(adj[u][i]);

    s.push(u);
}

void dfs2(int u)
{
    if(coloredInv[u])
        return;

    coloredInv[u] = 1;

    for (int i=0;i<adjInv[u].size();i++)
        dfs2(adjInv[u][i]);

    scc[u] = counter;
}

void _2sat(int n, int m, int a[], int b[])
{
    // adding edges to the graph
    for(int i=0;i<m;i++)
    {
        // variable x is mapped to x
        // variable -x is mapped to n+x = n-(-x)

        // for a[i] or b[i], addEdges -a[i] -> b[i]
        // AND -b[i] -> a[i]
        if (a[i]>0 && b[i]>0)
        {
            addEdges(a[i]+n, b[i]);
            addEdgesInverse(a[i]+n, b[i]);
            addEdges(b[i]+n, a[i]);
            addEdgesInverse(b[i]+n, a[i]);
        }

        else if (a[i]>0 && b[i]<0)
        {
            addEdges(a[i]+n, n-b[i]);
            addEdgesInverse(a[i]+n, n-b[i]);
            addEdges(-b[i], a[i]);
            addEdgesInverse(-b[i], a[i]);
        }

        else if (a[i]<0 && b[i]>0)
        {
            addEdges(-a[i], b[i]);
            addEdgesInverse(-a[i], b[i]);
            addEdges(b[i]+n, n-a[i]);
            addEdgesInverse(b[i]+n, n-a[i]);
        }

        else
        {
            addEdges(-a[i], n-b[i]);
            addEdgesInverse(-a[i], n-b[i]);
            addEdges(-b[i], n-a[i]);
            addEdgesInverse(-b[i], n-a[i]);
        }
    }

    // STEP 1 of Kosaraju's Algorithm which
    // traverses the original graph
    for (int i=1;i<=2*n;i++)
        if (!colored[i])
            dfs1(i);

    // STEP 2 of Kosaraju's Algorithm which
    // traverses the inverse graph. After this,
    // array scc[] stores the corresponding value
    while (!s.empty())
    {
        int n = s.top();
        s.pop();

        if (!coloredInv[n])
        {
            dfs2(n);
            counter++;
        }
    }

    for (int i=1;i<=n;i++)
    {
        // for any 2 variable x and -x lie in
        // same SCC
        if(scc[i]==scc[i+n])
        {
            cout << "The given expression "
                    "is unsatisfiable." << endl;
            return;
        }
    }

    // no such variables x and -x exist which lie
    // in same SCC
    cout << "The given expression is satisfiable."
         << endl;
    return;
}
```



```cpp
int main(){

    int n = 3, m = 2;

    int a[] = {1, -1, 1, -1};
    int b[] = {2, 2, -2, -2};

    _2sat(n, m, a, b);

    return 0;
}
```



```
"C:\Users\win 10\Documents\CLION\pruebas\cmake-build-debug\pruebas.exe"
The given expression is satisfiable.

Process finished with exit code 0
```

Build finished in 3 sec, 754 ms (moments ago)

5:1 CRLF UTF-8 4 spaces C++: pruebas | Debug Material Darker