Task-1: Graphs Graph A and Graph B – apply Prim's algorithms to find the Minimum Spanning Tree (MST).

Prims.cpp - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global>

Start here | kru.cpp | Prims.cpp

```cpp
#include<bits/stdc++.h>
#define pii pair<int, int>

using namespace std;
void prims(unordered_map<int, vector<pii>> &adj, int n, int st)
{
    vector <int> key(n, INT_MAX);
    vector <int> parrent(n, -1);
    vector <bool> mst(n, false);
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    key[st]=0;
    pq.push({key[st],st});
    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        if(mst[u]) continue;
        mst[u] = true;
        for(auto &edge: adj[u])
        {
            int v = edge.first;
            int wt = edge.second;
            if (!mst[v] and wt <key[v])
            {
                key[v]=wt;
                parrent[v]=u;
                pq.push({key[v],v});
            }
        }
    }
    int mcost=0;
```

G:\3rd year 1st semester\Practical Lab\Algorithm Design and Analysis (sessional)\after ... | C/C++ | Windows (CR+LF) | WINDOWS-1252 | Line 1, Col 1, Pos 0 | Insert | Read/Write | default

Search | ENG INTL | 12:15 AM 12/30/2024

Prims.cpp - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

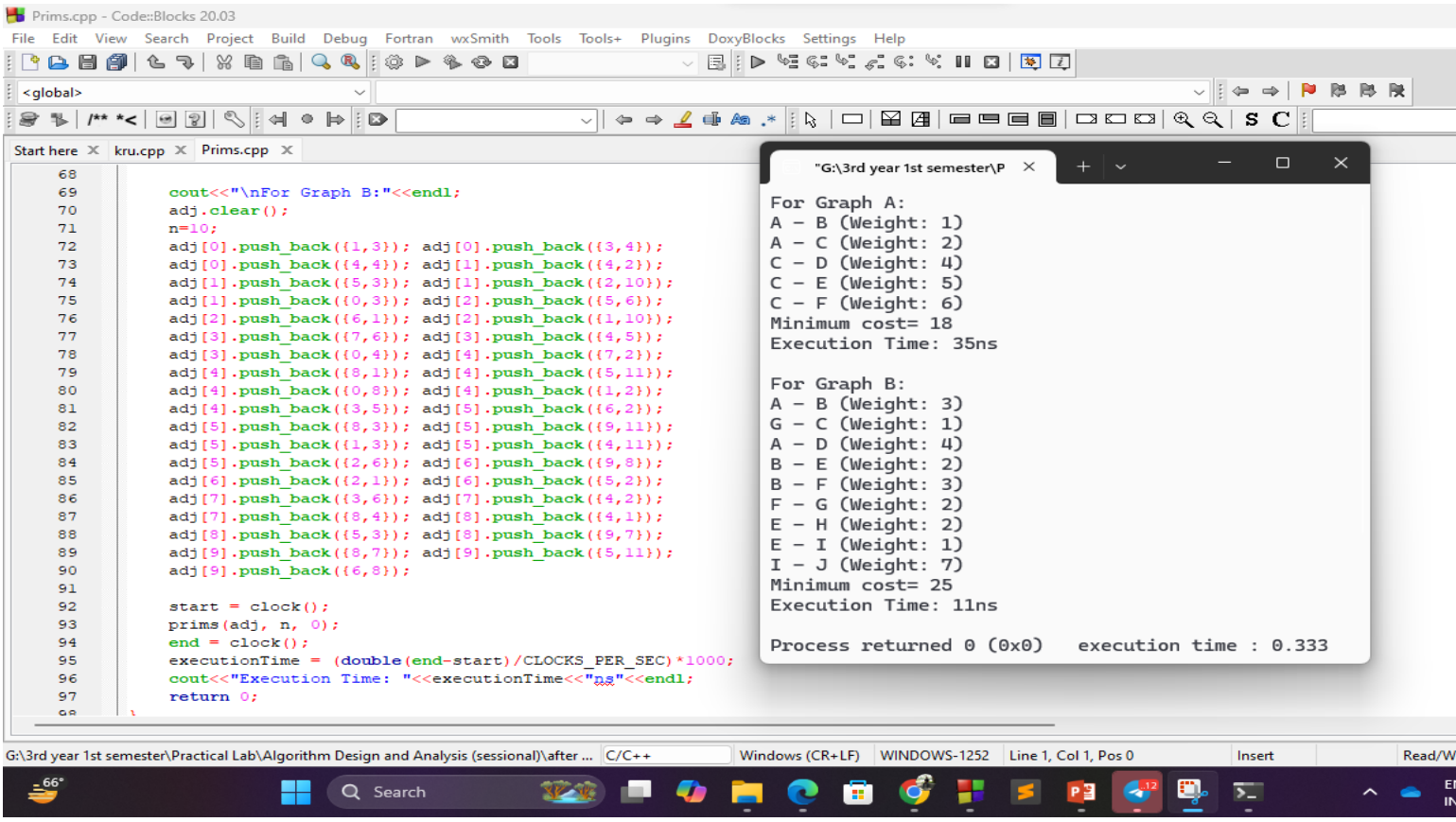
<global>

Start here | kru.cpp | Prims.cpp

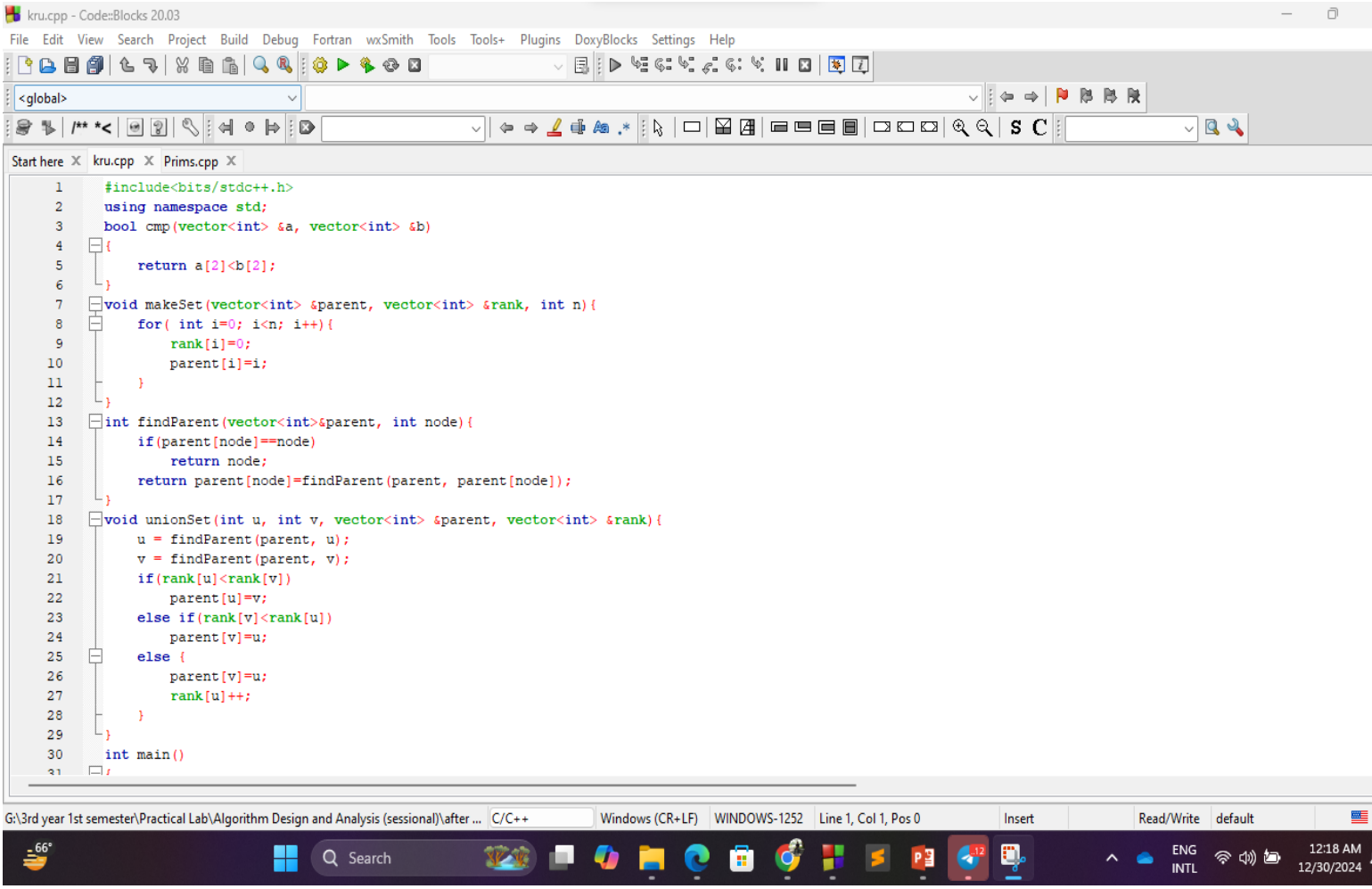
```cpp
    char Ar[] = {'A','B','C','D','E','F','G','H','I','J'};
    for(int i=0; i<n; i++){
        if(parrent[i] != -1){
            mcost += key[i];
            cout<< Ar[parrent[i]] <<" - "<< Ar[i] << " (Weight: "<<key[i] <<")\n";
        }
    }
    cout<<"Minimum cost= "<<mcost<<endl;
}

int main(){
    int n=6;
    unordered_map<int, vector<pair<int, int>>> adj;
    adj[0].push_back({1, 1}); // A-B
    adj[0].push_back({2, 2}); // A-C
    adj[1].push_back({0, 1}); // B-A
    adj[1].push_back({2, 3}); // B-C
    adj[2].push_back({0, 2}); // C-A
    adj[2].push_back({1, 3}); // C-B
    adj[2].push_back({3, 4}); // C-D
    adj[2].push_back({4, 5}); // C-E
    adj[2].push_back({5, 6}); // C-F
    adj[3].push_back({2, 4}); // D-C
    adj[3].push_back({4, 7}); // D-E
    adj[4].push_back({2, 4}); // E-C
    adj[4].push_back({3, 7}); // E-D
    adj[4].push_back({5, 8}); // E-F
    adj[5].push_back({2, 6}); // F-C
    adj[5].push_back({4, 8}); // F-E
    adj[9].push_back({6,8});
```

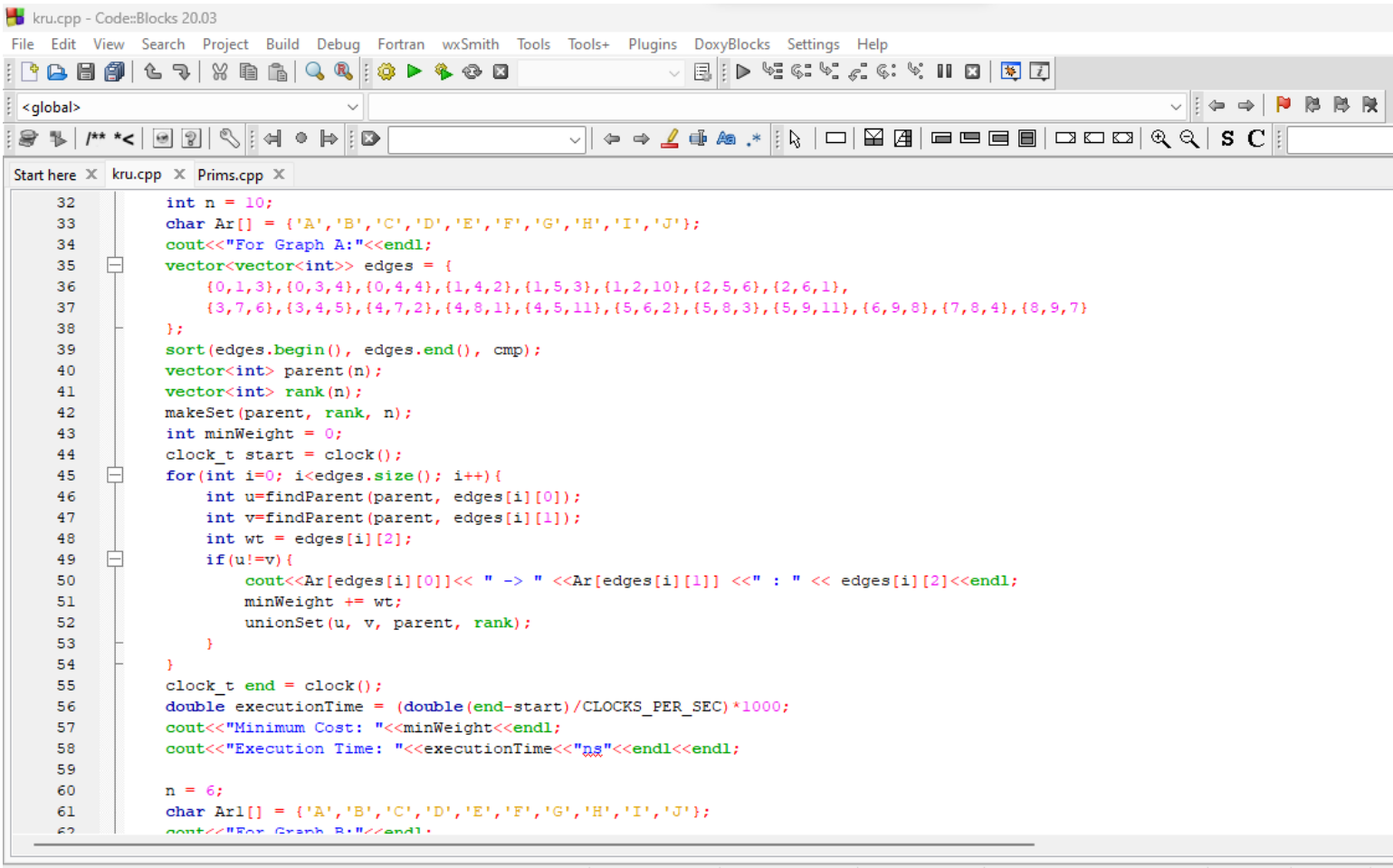
G:\3rd year 1st semester\Practical Lab\Algorithm Design and Analysis (sessional)\after ... | C/C++ | Windows (CR+LF) | WINDOWS-1252 | Line 1, Col 1, Pos 0 | Insert

Search

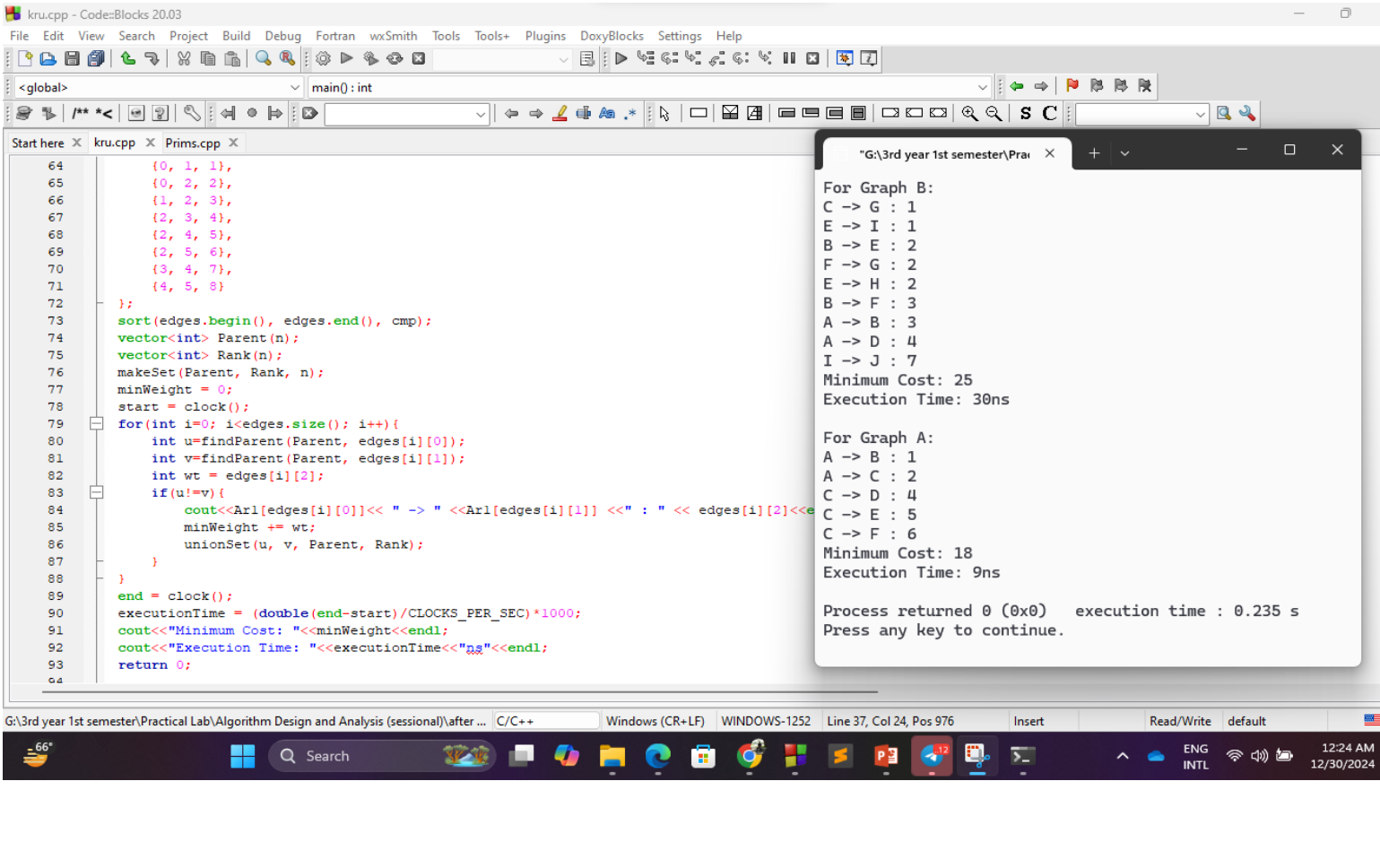


Task-2: Graphs Graph A and Graph B – apply Kruskal's algorithms to find the Minimum Spanning Tree (MST).





```
    int n = 10;
    char Ar[] = {'A','B','C','D','E','F','G','H','I','J'};
    cout<<"For Graph A:"<<endl;
    vector<vector<int>> edges = {
        {0,1,3},{0,3,4},{0,4,4},{1,4,2},{1,5,3},{1,2,10},{2,5,6},{2,6,1},
        {3,7,6},{3,4,5},{4,7,2},{4,8,1},{4,5,11},{5,6,2},{5,8,3},{5,9,11},{6,9,8},{7,8,4},{8,9,7}
    };
    sort(edges.begin(), edges.end(), cmp);
    vector<int> parent(n);
    vector<int> rank(n);
    makeSet(parent, rank, n);
    int minWeight = 0;
    clock_t start = clock();
    for(int i=0; i<edges.size(); i++){
        int u=findParent(parent, edges[i][0]);
        int v=findParent(parent, edges[i][1]);
        int wt = edges[i][2];
        if(u!=v){
            cout<<Ar[edges[i][0]]<< " -> " <<Ar[edges[i][1]] <<" : " << edges[i][2]<<endl;
            minWeight += wt;
            unionSet(u, v, parent, rank);
        }
    }
    clock_t end = clock();
    double executionTime = (double(end-start)/CLOCKS_PER_SEC)*1000;
    cout<<"Minimum Cost: "<<minWeight<<endl;
    cout<<"Execution Time: "<<executionTime<<"ns"<<endl<<endl;

    n = 6;
    char Arl[] = {'A','B','C','D','E','F','G','H','I','J'};
```



```
        {0, 1, 1},
        {0, 2, 2},
        {1, 2, 3},
        {2, 3, 4},
        {2, 4, 5},
        {2, 5, 6},
        {3, 4, 7},
        {4, 5, 8}
    };
    sort(edges.begin(), edges.end(), cmp);
    vector<int> Parent(n);
    vector<int> Rank(n);
    makeSet(Parent, Rank, n);
    minWeight = 0;
    start = clock();
    for(int i=0; i<edges.size(); i++){
        int u=findParent(Parent, edges[i][0]);
        int v=findParent(Parent, edges[i][1]);
        int wt = edges[i][2];
        if(u!=v){
            cout<<Arl[edges[i][0]]<< " -> " <<Arl[edges[i][1]] <<" : " << edges[i][2]<<e
            minWeight += wt;
            unionSet(u, v, Parent, Rank);
        }
    }
    end = clock();
    executionTime = (double(end-start)/CLOCKS_PER_SEC)*1000;
    cout<<"Minimum Cost: "<<minWeight<<endl;
    cout<<"Execution Time: "<<executionTime<<"ns"<<endl;
    return 0;
```

```
"G:\3rd year 1st semester\Prac
For Graph B:
C -> G : 1
E -> I : 1
B -> E : 2
F -> G : 2
E -> H : 2
B -> F : 3
A -> B : 3
A -> D : 4
I -> J : 7
Minimum Cost: 25
Execution Time: 30ns

For Graph A:
A -> B : 1
A -> C : 2
C -> D : 4
C -> E : 5
C -> F : 6
Minimum Cost: 18
Execution Time: 9ns

Process returned 0 (0x0)   execution time : 0.235 s
Press any key to continue.
```