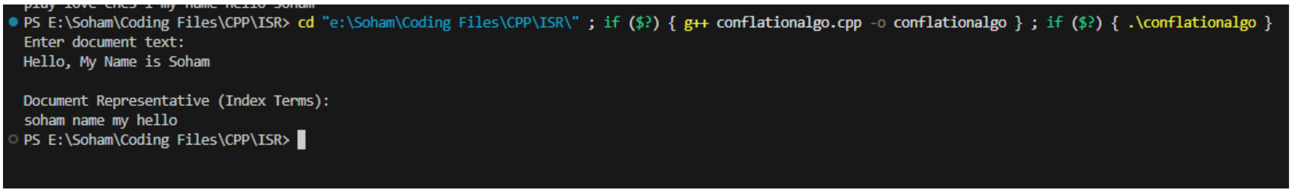
01 OUTPUT

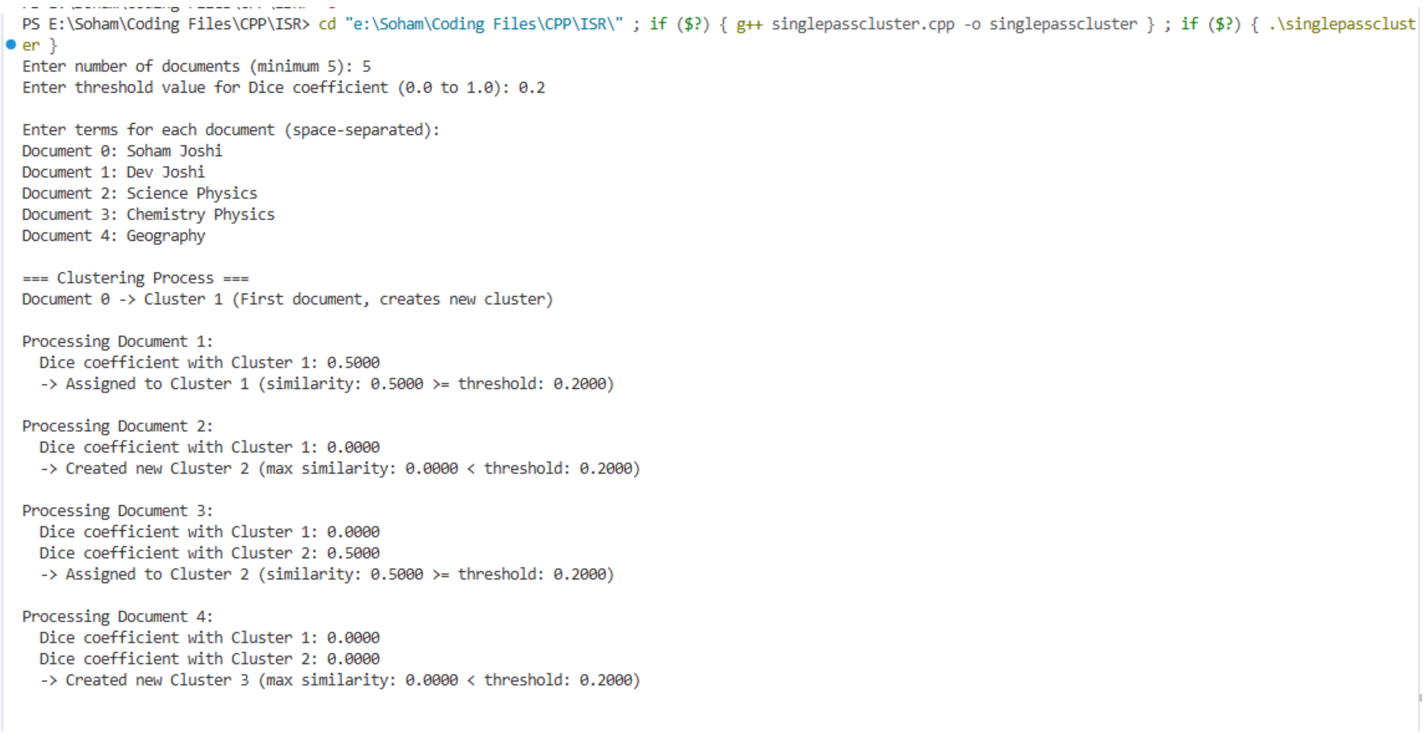
```
#include <bits/stdc++.h>
using namespace std;
unordered_set<string> stopwords = {
    "the", "is", "at", "of", "on", "and", "a", "to", "in", "it", "for", "this", "that", "an", "by", "as"
};
vector<string> suffixes = {"ingly", "edly", "ing", "ed", "es", "ly", "al", "s"};
map<string, string> equivalentStems = {
    {"absorpt", "absorb"},  // absorpt → absorb
    {"analys", "analyz"}    // analys → analyz
};
string toLowerCase(string s) {
    for (size_t i = 0; i < s.size(); i++)
        s[i] = tolower(s[i]);
    return s;
}
string cleanWord(const string &s) {
    string res;
    for (size_t i = 0; i < s.size(); i++) {
        if (isalpha(s[i]))
            res += tolower(s[i]);
    }
    return res;
}
bool endsWith(const string &word, const string &suffix) {
    if (word.size() < suffix.size()) return false;
    return word.compare(word.size() - suffix.size(), suffix.size(), suffix) == 0;
}
string stripSuffix(string word) {
    for (size_t i = 0; i < suffixes.size(); i++) {
        string suf = suffixes[i];
        if (word.size() > suf.size() + 2 && endsWith(word, suf)) {
            word = word.substr(0, word.size() - suf.size());
            break; // remove only one (longest) suffix
        }
    }
    return word;
}
string normalizeStem(string word) {
    if (equivalentStems.find(word) != equivalentStems.end())
        return equivalentStems[word];
    return word;
}

int main() {
    cout << "Enter document text:\n";
    string line;
    getline(cin, line);
    stringstream ss(line);
    string word;
    unordered_set<string> stems;
    while (ss >> word) {
        word = cleanWord(word);
        if (word.empty() || stopwords.count(word)) continue;

        word = stripSuffix(word);
        word = normalizeStem(word);
        stems.insert(word);
    }
    cout << "\nDocument Representative (Index Terms):\n";
    for (unordered_set<string>::iterator it = stems.begin(); it != stems.end(); ++it)
        cout << *it << " ";
    cout << endl;
    return 0;
}
```



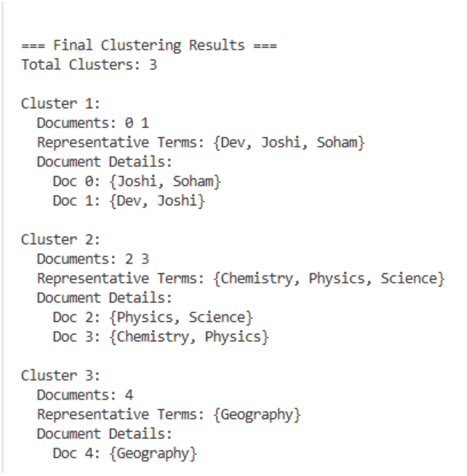
```
  play love ches i my name hello soham
● PS E:\Soham\Coding Files\CPP\ISR> cd "e:\Soham\Coding Files\CPP\ISR\" ; if ($?) { g++ conflationalgo.cpp -o conflationalgo } ; if ($?) { .\conflationalgo }
  Enter document text:
  Hello, My Name is Soham

  Document Representative (Index Terms):
  soham name my hello
○ PS E:\Soham\Coding Files\CPP\ISR> 
```

02 OUTPUT

```
#include <bits/stdc++.h>
using namespace std;
struct Document {
    int docId;
    set<string> terms;
};
struct Cluster {
    int clusterId;
    set<string> representative; // Cluster centroid (union of terms)
    vector<int> documentIds;
};
double calculateDiceCoefficient(const set<string>& set1, const set<string>& set2) {
    if (set1.empty() && set2.empty()) return 1.0;
    if (set1.empty() || set2.empty()) return 0.0;

    set<string> intersection;
    set_intersection(set1.begin(), set1.end(),
                     set2.begin(), set2.end(),
                     inserter(intersection, intersection.begin()));

    double dice = (2.0 * intersection.size()) / (set1.size() + set2.size());
    return dice;
}
void updateClusterRepresentative(Cluster& cluster, const vector<Document>& documents) {
    cluster.representative.clear();
    for (int docId : cluster.documentIds) {
        for (const string& term : documents[docId].terms) {
            cluster.representative.insert(term);
        }
    }
}

vector<Cluster> singlePassClustering(vector<Document>& documents, double threshold) {
    vector<Cluster> clusters;

    if (documents.empty()) return clusters;

    Cluster firstCluster;
    firstCluster.clusterId = 1;
    firstCluster.representative = documents[0].terms;
    firstCluster.documentIds.push_back(0);
    clusters.push_back(firstCluster);

    cout << "\n=== Clustering Process ===\n";
    cout << "Document 0 -> Cluster 1 (First document, creates new cluster)\n";

    for (size_t i = 1; i < documents.size(); i++) {
        bool assigned = false;
        int bestCluster = -1;
        double maxSimilarity = -1.0;

        cout << "\nProcessing Document " << i << ":\n";
        for (size_t j = 0; j < clusters.size(); j++) {
            double similarity = calculateDiceCoefficient(
                documents[i].terms,
                clusters[j].representative
            );
            cout << "  Dice coefficient with Cluster " << clusters[j].clusterId
                 << ": " << fixed << setprecision(4) << similarity << "\n";

            if (similarity > maxSimilarity) {
                maxSimilarity = similarity;
                bestCluster = j;
            }
        }
        if (maxSimilarity >= threshold) {
            clusters[bestCluster].documentIds.push_back(i);
            // Step 5: Update cluster representative
            updateClusterRepresentative(clusters[bestCluster], documents);
            assigned = true;
            cout << "  -> Assigned to Cluster " << clusters[bestCluster].clusterId
                 << " (similarity: " << maxSimilarity << " >= threshold: " << threshold << ")\n";
        }

        // Step 6: Create new cluster if no suitable cluster found
        if (!assigned) {
            Cluster newCluster;
            newCluster.clusterId = clusters.size() + 1;
            newCluster.representative = documents[i].terms;
            newCluster.documentIds.push_back(i);
            clusters.push_back(newCluster);
            cout << "  -> Created new Cluster " << newCluster.clusterId
                 << " (max similarity: " << maxSimilarity << " < threshold: " << threshold << ")\n";
        }
    }

    return clusters;
}
void displayClusters(const vector<Cluster>& clusters, const vector<Document>& documents) {
    cout << "\n\n=== Final Clustering Results ===\n";
    cout << "Total Clusters: " << clusters.size() << "\n\n";

    for (const Cluster& cluster : clusters) {
        cout << "Cluster " << cluster.clusterId << ":\n";
        cout << "  Documents: ";
        for (int docId : cluster.documentIds) {
            cout << docId << " ";
        }
        cout << "\n";

        cout << "  Representative Terms: {";
        bool first = true;
        for (const string& term : cluster.representative) {
            if (!first) cout << ", ";
            cout << term;
            first = false;
        }
        cout << "}\n";

        cout << "  Document Details:\n";
        for (int docId : cluster.documentIds) {
            cout << "    Doc " << docId << ": {";
            first = true;
            for (const string& term : documents[docId].terms) {
                if (!first) cout << ", ";
                cout << term;
                first = false;
            }
            cout << "}\n";
        }
        cout << "\n";
    }
}

int main() {
    int numDocuments;
    double threshold;

    cout << "Enter number of documents (minimum 5): ";
    cin >> numDocuments;

    if (numDocuments < 5) {
        cout << "Error: Minimum 5 documents required!\n";
        return 1;
    }

    cout << "Enter threshold value for Dice coefficient (0.0 to 1.0): ";
    cin >> threshold;

    if (threshold < 0.0 || threshold > 1.0) {
        cout << "Error: Threshold must be between 0.0 and 1.0!\n";
        return 1;
    }
    cin.ignore(); // Clear newline from buffer
    vector<Document> documents(numDocuments);
    cout << "\nEnter terms for each document (space-separated):\n";
    for (int i = 0; i < numDocuments; i++) {
        documents[i].docId = i;
        cout << "Document " << i << ": ";
        string line;
        getline(cin, line);
        stringstream ss(line);
        string term;
        while (ss >> term) {
            documents[i].terms.insert(term);
        }
        if (documents[i].terms.empty()) {
            cout << "Warning: Document " << i << " has no terms!\n";
        }
    }
    // Perform clustering
    vector<Cluster> clusters = singlePassClustering(documents, threshold);
    // Display results
    displayClusters(clusters, documents);
    return 0;
}
```



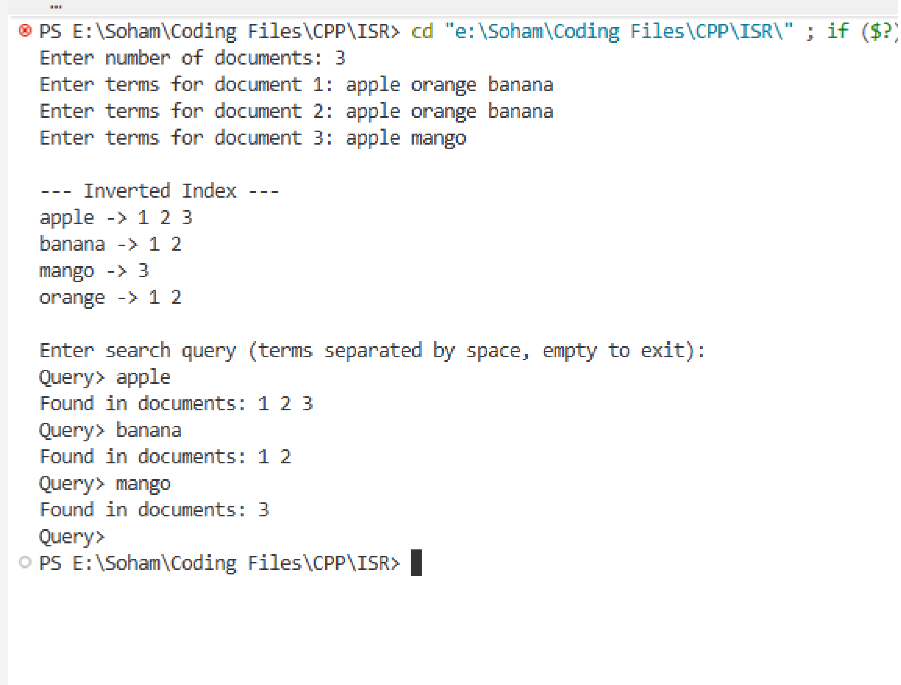
```
PS E:\Soham\Coding Files\CPP\ISR> cd "e:\Soham\Coding Files\CPP\ISR\" ; if ($?) { g++ singlepasscluster.cpp -o singlepasscluster } ; if ($?) { .\singlepassclust
er }
Enter number of documents (minimum 5): 5
Enter threshold value for Dice coefficient (0.0 to 1.0): 0.2

Enter terms for each document (space-separated):
Document 0: Soham Joshi
Document 1: Dev Joshi
Document 2: Science Physics
Document 3: Chemistry Physics
Document 4: Geography

=== Clustering Process ===
Document 0 -> Cluster 1 (First document, creates new cluster)

Processing Document 1:
  Dice coefficient with Cluster 1: 0.5000
  -> Assigned to Cluster 1 (similarity: 0.5000 >= threshold: 0.2000)

Processing Document 2:
  Dice coefficient with Cluster 1: 0.0000
  -> Created new Cluster 2 (max similarity: 0.0000 < threshold: 0.2000)

Processing Document 3:
  Dice coefficient with Cluster 1: 0.0000
  Dice coefficient with Cluster 2: 0.5000
  -> Assigned to Cluster 2 (similarity: 0.5000 >= threshold: 0.2000)

Processing Document 4:
  Dice coefficient with Cluster 1: 0.0000
  Dice coefficient with Cluster 2: 0.0000
  -> Created new Cluster 3 (max similarity: 0.0000 < threshold: 0.2000)
```



```
=== Final Clustering Results ===
Total Clusters: 3

Cluster 1:
  Documents: 0 1
  Representative Terms: {Dev, Joshi, Soham}
  Document Details:
    Doc 0: {Joshi, Soham}
    Doc 1: {Dev, Joshi}

Cluster 2:
  Documents: 2 3
  Representative Terms: {Chemistry, Physics, Science}
  Document Details:
    Doc 2: {Physics, Science}
    Doc 3: {Chemistry, Physics}

Cluster 3:
  Documents: 4
  Representative Terms: {Geography}
  Document Details:
    Doc 4: {Geography}
```

03 OUTPUT

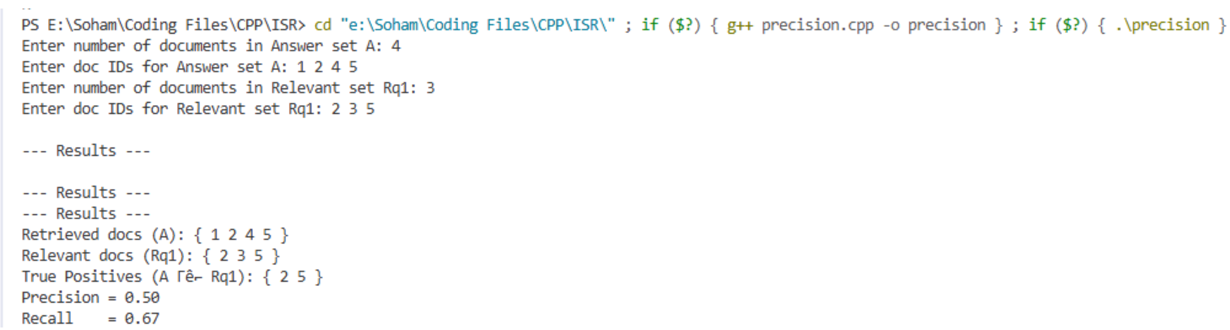
```
#include <bits/stdc++.h>
using namespace std;

// Utility: intersect two sorted vectors
vector<int> intersectVectors(const vector<int>& a, const vector<int>& b) {
    vector<int> result;
    int i = 0, j = 0;
    while (i < a.size() && j < b.size()) {
        if (a[i] == b[j]) {
            result.push_back(a[i]);
            i++; j++;
        } else if (a[i] < b[j]) {
            i++;
        } else {
            j++;
        }
    }
    return result;
}

int main() {
    map<string, vector<int>> invertedIndex;

    int n;
    cout << "Enter number of documents: ";
    cin >> n;
    cin.ignore();

    // Build inverted index
    for (int docId = 1; docId <= n; docId++) {
        cout << "Enter terms for document " << docId << ": ";
        string line;
        getline(cin, line);

        stringstream ss(line);
        string term;
        while (ss >> term) {
            // avoid duplicate docIds for a term
            if (invertedIndex[term].empty() || invertedIndex[term].back() != docId) {
                invertedIndex[term].push_back(docId);
            }
        }
    }

    cout << "\n--- Inverted Index ---\n";
    for (auto &entry : invertedIndex) {
        cout << entry.first << " -> ";
        for (int id : entry.second) cout << id << " ";
        cout << "\n";
    }

    // Searching
    cout << "\nEnter search query (terms separated by space, empty to exit):\n";
    string query;
    while (true) {
        cout << "Query> ";
        getline(cin, query);
        if (query.empty()) break;

        stringstream ss(query);
        string term;
        vector<int> result;
        bool first = true;

        while (ss >> term) {
            if (invertedIndex.find(term) == invertedIndex.end()) {
                result.clear(); // no docs for this term
                break;
            }

            if (first) {
                result = invertedIndex[term];
                first = false;
            } else {
                result = intersectVectors(result, invertedIndex[term]);
            }
        }

        if (result.empty()) {
            cout << "No documents found.\n";
        } else {
            cout << "Found in documents: ";
            for (int id : result) cout << id << " ";
            cout << "\n";
        }
    }

    return 0;
}
```



```
PS E:\Soham\Coding Files\CPP\ISR> cd "e:\Soham\Coding Files\CPP\ISR\" ; if ($?
Enter number of documents: 3
Enter terms for document 1: apple orange banana
Enter terms for document 2: apple orange banana
Enter terms for document 3: apple mango

--- Inverted Index ---
apple -> 1 2 3
banana -> 1 2
mango -> 3
orange -> 1 2

Enter search query (terms separated by space, empty to exit):
Query> apple
Found in documents: 1 2 3
Query> banana
Found in documents: 1 2
Query> mango
Found in documents: 3
Query>
PS E:\Soham\Coding Files\CPP\ISR>
```

04 OUTPUT

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int nA, nR;

  cout << "Enter number of documents in Answer set A: ";
  cin >> nA;
  set<int> A;
  cout << "Enter doc IDs for Answer set A: ";
  for (int i = 0; i < nA; i++) {
    int d; cin >> d;
    A.insert(d);
  }

  cout << "Enter number of documents in Relevant set Rq1: ";
  cin >> nR;
  set<int> R;
  cout << "Enter doc IDs for Relevant set Rq1: ";
  for (int i = 0; i < nR; i++) {
    int d; cin >> d;
    R.insert(d);
  }

  // Intersection: true positives
  vector<int> intersection;
  set_intersection(A.begin(), A.end(), R.begin(), R.end(),
back_inserter(intersection));

  int truePositives = intersection.size();
  double precision = (nA == 0) ? 0 : (double)truePositives / nA;
  double recall    = (nR == 0) ? 0 : (double)truePositives / nR;

  cout << "\n--- Results ---\n";
  cout << "Retrieved docs (A): { "; for (int x : A) cout << x << " "; cout << "}
\n";
  cout << "Relevant docs (Rq1): { "; for (int x : R) cout << x << " "; cout << "}
\n";
  cout << "True Positives (A ∩ Rq1): { "; for (int x : intersection) cout << x << "
"; cout << "}\n";

  cout << fixed << setprecision(2);
  cout << "Precision = " << precision << "\n";
  cout << "Recall    = " << recall << "\n";

  return 0;
}
```

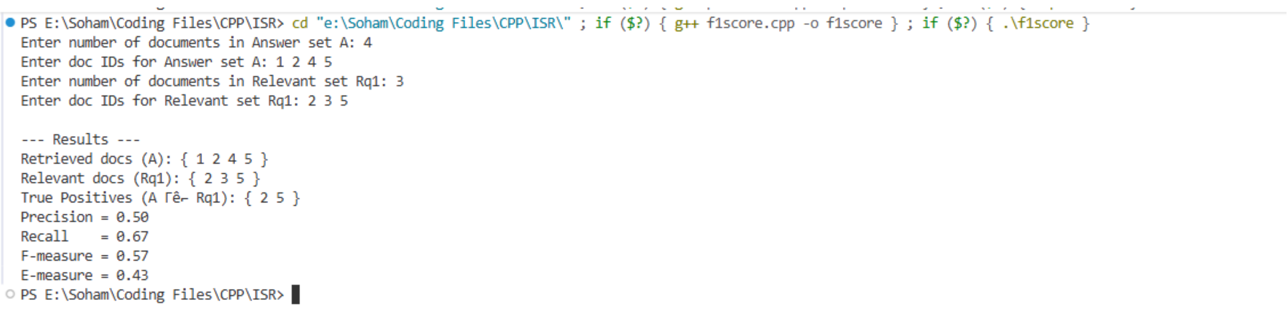


```
PS E:\Soham\Coding Files\CPP\ISR> cd "e:\Soham\Coding Files\CPP\ISR\" ; if ($?) { g++ precision.cpp -o precision } ; if ($?) { .\precision }
Enter number of documents in Answer set A: 4
Enter doc IDs for Answer set A: 1 2 4 5
Enter number of documents in Relevant set Rq1: 3
Enter doc IDs for Relevant set Rq1: 2 3 5

--- Results ---

--- Results ---
--- Results ---
Retrieved docs (A): { 1 2 4 5 }
Relevant docs (Rq1): { 2 3 5 }
True Positives (A Γê⌐ Rq1): { 2 5 }
Precision = 0.50
Recall    = 0.67
```

05 OUTPUT

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int nA, nR;

  cout << "Enter number of documents in Answer set A: ";
  cin >> nA;
  set<int> A;
  cout << "Enter doc IDs for Answer set A: ";
  for (int i = 0; i < nA; i++) {
    int d; cin >> d;
    A.insert(d);
  }

  cout << "Enter number of documents in Relevant set Rq1: ";
  cin >> nR;
  set<int> R;
  cout << "Enter doc IDs for Relevant set Rq1: ";
  for (int i = 0; i < nR; i++) {
    int d; cin >> d;
    R.insert(d);
  }

  // Intersection (True Positives)
  vector<int> intersection;
  set_intersection(A.begin(), A.end(), R.begin(), R.end(),
back_inserter(intersection));

  int truePositives = intersection.size();
  double precision = (nA == 0) ? 0 : (double)truePositives / nA;
  double recall    = (nR == 0) ? 0 : (double)truePositives / nR;

  // F-measure (harmonic mean)
  double fmeasure = (precision + recall == 0) ? 0 : (2 * precision * recall) /
(precision + recall);

  // E-measure with β = 1
  double beta = 1.0;
  double emeasure = (precision == 0 && recall == 0) ? 1 :
           1 - ((1 + beta * beta) * precision * recall) / (beta * beta * precision
+ recall);

  cout << "\n--- Results ---\n";
  cout << "Retrieved docs (A): { "; for (int x : A) cout << x << " "; cout << "}
\n";
```

```cpp
  cout << "Relevant docs (Rq1): { "; for (int x : R) cout << x << " "; cout << "}\n";
  cout << "True Positives (A ∩ Rq1): { "; for (int x : intersection) cout << x << " "; cout << "}\n";

  cout << fixed << setprecision(2);
  cout << "Precision = " << precision << "\n";
  cout << "Recall    = " << recall << "\n";
  cout << "F-measure = " << fmeasure << "\n";
  cout << "E-measure = " << emeasure << "\n";

  return 0;
}
```



```
● PS E:\Soham\Coding Files\CPP\ISR> cd "e:\Soham\Coding Files\CPP\ISR\" ; if ($?) { g++ f1score.cpp -o f1score } ; if ($?) { .\f1score }
  Enter number of documents in Answer set A: 4
  Enter doc IDs for Answer set A: 1 2 4 5
  Enter number of documents in Relevant set Rq1: 3
  Enter doc IDs for Relevant set Rq1: 2 3 5

  --- Results ---
  Retrieved docs (A): { 1 2 4 5 }
  Relevant docs (Rq1): { 2 3 5 }
  True Positives (A Γê⌐ Rq1): { 2 5 }
  Precision = 0.50
  Recall    = 0.67
  F-measure = 0.57
  E-measure = 0.43
○ PS E:\Soham\Coding Files\CPP\ISR> █
```