```cpp
#include <iostream>     // cout, endl
#include <fstream>      // fstream
#include <vector>
#include <algorithm>    // copy
#include <iterator>     // ostream_operator
#include<sstream>
#include<cstdlib>
#include<ostream>
#include <string>
#include <iomanip>
#include <random>
#include<cmath>
#include<utility>
#include <boost/tokenizer.hpp>
#include <boost/accumulators/accumulators.hpp>
#include <boost/accumulators/statistics.hpp>
#include <boost/bind.hpp>
#include <boost/ref.hpp>

#define SAMPLESIIZE 8
#define TOTALSIZE 60
#define N1EX 2
#define TOTALSAMPLE 4


typedef std::vector<double> DVector;
typedef std::vector<std::string> ColHead;
typedef std::vector<double> TempVec;
typedef std::vector<int> N1delns;
typedef std::vector<int> N2delns;
typedef std::vector<std::pair<std::string,double>> GeneMap;

using namespace boost::accumulators;
 typedef accumulator_set<double, stats<tag::mean, tag::variance>> accSet;
 double differentiation(std::vector<double> genedata,int group1,int group2);
 double ttest(accSet acc1,accSet acc2,int n1,int n2);

struct sort_pred {
    bool operator()(const std::pair<std::string,double> &left, const std::pair<std::string,double> &right)
{
        return left.second < right.second;
    }
};



int main()
{
    using namespace std;
    using namespace boost;

    string data("NCI60.csv");

    ifstream in(data.c_str());
    if (!in.is_open()) return 1;
    std::filebuf fb;
    fb.open ("test.txt",std::ios::out);
    std::ostream os(&fb);

    typedef tokenizer< escaped_list_separator<char> > Tokenizer;

    DVector vec;
    TempVec tempv;
    ColHead colhead;
    N1delns n1deln;
    N2delns n2deln;
    GeneMap gene;

    string line;
    int n=0;
    int a=0;

    while (getline(in,line))
    {

    if(n==0){
    Tokenizer firstline(line);
     Tokenizer::const_iterator token_iterator = firstline.begin();
    std::istringstream to_string;

    for(token_iterator;token_iterator!=firstline.end();token_iterator++)
    {
        string datum;

        to_string.clear();
        to_string.str(*token_iterator);
        to_string >> datum;
        colhead.push_back(datum);
}

    }

else{

    Tokenizer tok(line);
    Tokenizer::const_iterator token_iterator = tok.begin();



    std::istringstream to_double;

    std::string id(*token_iterator);


    for(++token_iterator; token_iterator != tok.end(); ++token_iterator)
    {
        int m=0;
       double datum = 0.0;

        to_double.clear();
        to_double.str(*token_iterator);
        to_double >> datum;
        tempv.push_back(datum);



        if(!to_double)
           continue;

        vec.push_back(datum);

    }
    int n1del=0;
    int n2del=0;
    for(int i=0;i<SAMPLESIIZE;i++)
        {
         if(tempv[i]==fabs(0))
           n1del++;

        }

        double dscore= differentiation(vec,SAMPLESIIZE-n1del,vec.size()+n1del-SAMPLESIIZE);
```

```
            gene.push_back(make_pair(id,dscore));

  os << gene[n-1].first<<","<<gene[n-1].second<<endl;
            //cout<<gene[n-1].first<<"\t"<<gene[n-1].second<<endl;
             tempv.clear();

       vec.clear();

    }
//cout<<gene.size()<<endl;
n++;
    }


    std::sort(gene.begin(),gene.end(),
              boost::bind(&std::pair<std::string, double>::second, _1) >
          boost::bind(&std::pair<std::string, double>::second, _2));
    //cout<<gene.size()<<endl;
    for (GeneMap::iterator it = gene.begin(); it != gene.begin()+20; ++it)

        cout << it->first <<","<< it->second << endl;


}


double differentiation(std::vector<double> genedata,int group1,int group2)
{
    accSet acc1;
    accSet acc2;
    double dscore;
    std::for_each(genedata.begin(), genedata.begin()+group1 ,
                                boost::bind<void>(boost::ref(acc1), _1));


    std::for_each(genedata.begin()+group1, genedata.end() ,
                                boost::bind<void>(boost::ref(acc2), _1));


   double tSamp = ttest(acc1,acc2,group1,group2);

    accSet randacc1;
    accSet randacc2;
    accSet randTest;
    for(int i=0;i<100;i++)
    {


    std::random_shuffle(genedata.begin(), genedata.end());

    std::for_each(genedata.begin(), genedata.begin()+group1 ,
                                boost::bind<void>(boost::ref(randacc1), _1));

    std::for_each(genedata.begin()+group1, genedata.end() ,
                                boost::bind<void>(boost::ref(randacc2), _1));


double tscore= ttest(randacc1,randacc2,group1,group2);
randTest(tscore);

    }
dscore = fabs((double)tSamp - (double)mean(randTest)) /
        sqrt(variance(randTest));


return dscore;
```

```
}


double ttest(accSet acc1,accSet acc2,int n1,int n2)
{

double mean1 = mean(acc1);
double mean2 = mean(acc2);
double variance1 = variance(acc1);
double variance2 = variance(acc2);
double tscore= ((mean2-mean1)/(sqrt(((variance1*variance1)/(double)n1)+((variance2*variance2)/(double)n2))));

return tscore;
}

```