1. What do you know about chisquare Test?
   - A chi-squared test, also referred to as χ^2 test (or chi-square test), is any statistical hypothesis test in which the sampling distribution of the test statistic is a chi-square distribution when the null hypothesis is true.
   - Pearson's chi-square test, also known as the chi-square goodness-of-fit test or chi-square test for independence. Pearson's chi-squared test ($\chi 2$) is a statistical test applied to sets of categorical data to evaluate how likely it is that any observed difference between the sets arose by chance.
   - Other chisquare tests include
     - CMH test: for stratified 2 by 2 tables to test whether rows and cols are independent
     - McNemar test for paired 2 by 2 table to test marginal homogeneity
     - Likelihood Ratio Test: test whether two nested models are equally well-fitted for the data
     - Ljung–Box test: in time series analysis, test whether any group of autocorrelation among residuals are different from 0.

3. X, Y are iid N(0,1) calculate p(X | X+Y > 0), try not use density function of joint distribution.
   - $P(X \mid X+Y > 0) = P(X+Y > 0 \mid X) \times P(X) / P(X+Y > 0) = 2P(Y > -x)\, P(X=x) = 2\Phi(x)\, f(x)$ by conditional probability and symmetry.

12. Implement the interface for matrix class in C++

```
#ifndef _MATRIX_H
#define _MATRIX_H
#include <vector>
template <typename T> class Matrix {
private:
 std::vector<std::vector<T> > mat;
 unsigned rows;
 unsigned cols;
public:
 Matrix(unsigned _rows, unsigned _cols, const T& _initial);
 Matrix(const Matrix<T>& rhs);
 virtual ~Matrix();
 // Operator overloading, for "standard" mathematical matrix
operations

 Matrix<T>& operator=(const Matrix<T>& rhs);
 // Matrix mathematical
operations

 Matrix<T> operator+(const Matrix<T>& rhs);
 Matrix<T>& operator+=(const Matrix<T>& rhs);
 Matrix<T> operator-(const Matrix<T>& rhs);
 Matrix<T>& operator-=(const Matrix<T>& rhs);
 Matrix<T> operator*(const Matrix<T>& rhs);
 Matrix<T>& operator*=(const Matrix<T>& rhs);
 Matrix<T> transpose();
 // Matrix/scalar
operations

 Matrix<T> operator+(const T& rhs);
 Matrix<T> operator-(const T& rhs);
 Matrix<T> operator*(const T& rhs);
 Matrix<T> operator/(const T& rhs);
 // Matrix/vector
operations

 std::vector<T> operator*(const std::vector<T>& rhs);
 std::vector<T> diag_vec();
 // Access the individual
elements

 T& operator()(const unsigned& row, const unsigned& col);
 const T& operator()(const unsigned& row, const unsigned& col) const;
 // Access the row and column
sizes

 unsigned get_rows() const;
 unsigned get_cols() const;
};
#endif
```

15. Given a string, return the longest palindromic subsequences

link: http://articles.leetcode.com/2011/11/longest-palindromic-substring-part-i.html

```
string longestPalindromeDP(string s) {
 int n = s.length();
 int longestBegin = 0;
 int maxLen = 1;
 bool table[1000][1000] = {false};
 for (int i = 0; i < n; i++) {
   table[i][i] = true;
 }
 for (int i = 0; i < n-1; i++) {
   if (s[i] == s[i+1]) {
     table[i][i+1] = true;
     longestBegin = i;
     maxLen = 2;
   }
 }
 for (int len = 3; len <= n; len++) {
   for (int i = 0; i < n-len+1; i++) {
     int j = i+len-1;
     if (s[i] == s[j] && table[i+1][j-1]) {
       table[i][j] = true;
       longestBegin = i;
       maxLen = len;
     }
   }
 }
 return s.substr(longestBegin, maxLen);
}
```