Q1.

The (Pearson's) chi-squared test 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. It is used to assess two types of comparison:

(1) A test of goodness of fit: establishes whether or not an observed frequency distribution differs from a theoretical distribution.

(2) A test of independence: assesses whether unpaired observations on two variables, expressed in a contingency table, are independent of each other.



<https://en.wikipedia.org/wiki/Chi-squared_test>

<https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test>

<http://stattrek.com/chi-square-test/independence.aspx?Tutorial=AP>

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda35f.htm>

Q11.

C++ Code:

**#include** <iostream>

**#include** <sstream>

**using** **namespace** std;

**struct** Node

{

**int** data;

Node \* next;

**Node**() { data = 0; next = NULL; }

**Node**(**int** d) { data = d; next = NULL; }

**Node**(**int** d, Node \* n) { data = d; next = n; }

};

**typedef** Node \* ListType;

**int** listLength = 0;

**void** **buildList**(ListType & theList)

{

string lineStr;

**getline**(cin, lineStr);

**if** (lineStr.empty())

{

theList = NULL;

**return**;

}

istringstream istr(lineStr);

ListType last = NULL;

**int** i;

istr >> i; // first one is a special case

theList = **new** Node(i);

listLength++;

last = theList;

**while** (istr >> i) { // comes out false if we reach end of string (i.e., eoln)

last->next = **new** Node(i);

last = last->next;

listLength++;

}

}

**void** **printList**(ListType & theList)

{

ListType p = theList;

cout << "The list: ";

**if** (p == NULL) { cout << "<Empty>" << **endl**; **return**; }

**while** (p != NULL)

{

cout << p->data << " ";

p = p->next;

}

cout << **endl**;

}

/\*

\* Method 1: consider three situations,

\* remove first, last, middle element.

void removeElmt(ListType & list, int loc)

{

if (list == NULL)

{ cout << "NULL!" << endl; return; }

if (loc > listLength)

{ cout << "Exceed!" << endl; return; }

ListType p = list;

if (loc == 1)

{

list = list->next;

p->next = NULL;

delete p;

}

for (int i = 1; i < loc-1; i++)

{ p = p->next; }

ListType temp = p->next;

if (loc == listLength)

{ p->next = NULL; }

else

{

p->next = p->next->next;

temp->next = NULL;

}

delete temp;

return;

}

\*/

/\*

\* Method 2: Use 2-level pointer

\* Hold a pointer to the pointer that points to Node,

\* that is we use a double indirection.

\* This enables us to both modify the pointer to the Node

\* and to modify the record without tracking the previous Node.

\* We don't need to special case dealing with the first node.

\*/

**void** **removeElmt**(ListType \* head, **int** loc) {

**if** (head == NULL || loc > listLength) { **return**; }

ListType \* current = head;

**for** (**int** i = 1; i < loc; i++)

{ current = &(\*current)->next; }

ListType next = (\*current)->next;

**delete** \*current;

\*current = next;

**return**;

}

**int** **main**(**int** argc, **char** \* argv[])

{

ListType lst;

cout << "Please input list: ";

buildList(lst);

printList(lst);

removeElmt(&lst, 5);

printList(lst);

**return** 0;

}

<http://www.algolist.net/Data_structures/Singly-linked_list/Removal>

<https://www.cs.bu.edu/teaching/cs112/spring-2000/linked-list-delete/>

<http://codereview.stackexchange.com/questions/496/whats-the-cleanest-way-to-implement-a-delete-from-a-singly-linked-list-in-c>

Q12.

C++ Code:

**#include** <iostream>

**using** **namespace** std;

**class** matrix

{

**int** \*\*p, m, n;

**public**:

**matrix**(**int** row, **int** col)

{

m = row;

n = col;

p = **new** **int**\*[m];

**for** (**int** i = 0; i < m; ++i)

{

p[i] = **new** **int**[n];

**for** (**int** j = 0; j < n; ++j)

p[i][j] = 0;

}

}

**void** **accept**()

{

cout<<"Enter matrix elements: ";

**for**(**int** i = 0; i < m; i++)

{

**for**(**int** j = 0; j < n; j++)

{

cin >> p[i][j];

}

}

}

**void** **display**()

{

cout <<"The matrix is:\n";

**for**(**int** i = 0; i < m; i++)

{

**for**(**int** j = 0; j < n; j++)

{

cout << p[i][j] <<" ";

}

cout <<**endl**;

}

}

matrix& **operator+** (matrix& lhs, **const** matrix& m1)

{

**return** (\***this** += m1);

}

matrix& **operator\*** (**const** matrix& m1)

{

**return** (\***this** \*= m1);

}

matrix& **operator+=** (**const** matrix& rhs)

{

**for**(**int** i = 0; i < m; i++)

{

**for**(**int** j = 0; j < n; j++)

{

p[i][j] += rhs.p[i][j];

}

}

**return** \***this**;

}

matrix& **operator\*=** (**const** matrix& T)

{

**if**(n == T.m)

{

**for**(**int** i = 0; i < T.m; ++i)

{

**for**(**int** k = 0; k < n; ++k)

{

p[i][k] \*= T.p[k][i];

}

}

}

**return** \***this**;

}

matrix& **operator=** (**const** matrix& T)

{

p = T.p;

n = T.n;

m = T.m;

**return** \***this**;

}

};

Q5.

<http://math.stackexchange.com/questions/13959/if-a-1-meter-rope-is-cut-at-two-uniformly-randomly-chosen-points-what-is-the-av>