# Diskret matematik

# Programmeringslaboration

## Kombinatorik och sannolikhet

Jenny Söderberg 890728-6648

Alexander Milton 940510-8136

Ht2014

### Uppgift 1: Tärningar

Räkna ut sannolikheten P(E|F) för n kast med en m-sidig tärning, där

E = ”alla sidor på tärningen är med i följden av tärningstest”,

F = ”följden av tärningskast är växande”.

*a)*

Räkna ut P(E), P(F) och P(E|F) för (m,n) {(6,8),(6,6),(4,5),(4,10),(3,10),(2,10)}

#### Omega

integer n tärningskast

integer m sidor

integer[][] Omega( n, m )

{

integer[][] omega

integer i, j

for( 0 < i < m^n)

{

for( 0 < j < n )

{

omega[i][j] = (i/( m^j ) %m )+1

}

}

return omega

}

#### E

integer n tärningskast

integer m sidor

integer[][] E( n, m )

{

integer[][] e

integer i, die

boolean allValues, set to True

for( 0 < i < omega.size )

{

for( 1 < die <= m )

{

if ( omega[i] does not contain die )

{

allValues = false

}

}

if( allValues = true )

{

put omega[i] in e

}

}

return e

}

#### F

integer n tärningskast

integer m sidor

integer[][] F( n, m )

{

integer n tärningskast

integer m sidor

integer[][] f

integer i, j

boolean lessThan, set to False

for ( 0 < i < omega.size )

{

for ( 1 < j < omega[i].size )

{

if ( omega[i][j] < omega[i][j - 1] )

{

lessThan = true

}

}

if ( lessThan = false )

{

put omega[i] in f

}

}

return f

}

#### Section

integer[][] A, B

integer[][] section( A, B )

{

integer[][] sect

integer i, j

for ( 0 < i < A.size )

{

for ( 0 < j < B.size )

{

if ( A[i] = B[i] )

{

put A[i] in sect

}

}

}

return sect

}

#### Calculate probability 1a

integer n tärningskast

integer m sidor

Input: (m,n) {(6,8),(6,6),(4,5),(4,10),(3,10),(2,10)}

E = E(n,m)

F = F(n,m)

E∩F = section(E,F)

Print “P(E): ” E.size/omega.size

Print “P(F): “ F.size/omega.size

Print “P(E|F): “ E∩F.size/F.size

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **m** | **n** | **P(E)** | **P(F)** | **P(E|F)** |
| 6 | 8 | 0.114026 | 0.000766247 | 0.016317 |
| 6 | 6 | 0.0154321 | 0.00990226 | 0.0021645 |
| 4 | 5 | 0.234375 | 0.0546875 | 0.0714286 |
| 4 | 10 | 0.780602 | 0.000272751 | 0.293706 |
| 3 | 10 | 0.948026 | 0.00111772 | 0.545455 |
| 2 | 10 | 0.998047 | 0.0107422 | 0.818182 |

#### Slutsats:

P(E|F) är sannolikheten för att alla tärningsvärden finns med om värdena i följden är stigande.

b)

Undersök den ungefärliga sannolikheten P(E|F) när experimentet görs x gånger.

integer n tärningskast

integer m sidor

integer x gånger

#### Kasta tärning

integer kasta\_tarning( m )

{

if ( m < 2 )

{

return NULL

}

integer value = randomValue % m

return value

}

#### Kasta tärningar n gånger

integer[] kasta\_tarningar( n, m )

{

integer[] values

for ( 0 < i < n )

{

put ( kasta\_tarning( m ) ) in values

}

return values

}

#### Calculate probability 1b

integer n, m, x

oneBprobability( n, m, x )

{

integer[][] values

for ( 0 < i < x )

{

int[] tempValues = kasta\_tarningar( n, m )

put tempValues in values

}

omega = values

integer b = F( n, m ).size

integer a = section( E( n, m ), F( n, m ).size )

if (b ≠ 0)

{

float probability = a / b

print “P(E|F): ” probability

}

}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **m** | **n** | **X = 1000** | **X = 10^4** | **X=10^5** | **P(E|F)** |
| 6 | 8 | 0 | 0 | 0.0229885 | 0.016317 |
| 6 | 6 | 0 | 0 | 0.00391389 | 0.0021645 |
| 4 | 5 | 0.0444444 | 0.0597826 | 0.0706121 | 0.0714286 |
| 4 | 10 | 0 | 0.333333 | 0.318182 | 0.293706 |
| 3 | 10 | 0 | 0.428571 | 0.5 | 0.545455 |
| 2 | 10 | 0.75 | 0.821053 | 0.840156 | 0.818182 |

c)

n≥m

### Uppgift 2: Binomialkoefficienter

a)

n är radindex i Pascals triangel, k är index för elementet i en rad.

#### Binomialkoefficienter

integer n, k

integer binom\_pascal( n, k )

{

if ( k > n )

{

return 0

}

integer[][] pascalVector

for ( 0 < i <= n ) // Row number

{

integer[] rowVector

for ( 0 < j <= i ) // Element in row

{

if ( j = 0 or j = i )

{

put 1 in rowVector

print 1

}

else

{

integer firstValue = pascalVector[i – 1] [j – 1]

integer secondValue = pascalVector[i - 1][j]

print (firstValue + secondValue)

put (firstValue + secondValue) in rowVector

}

}

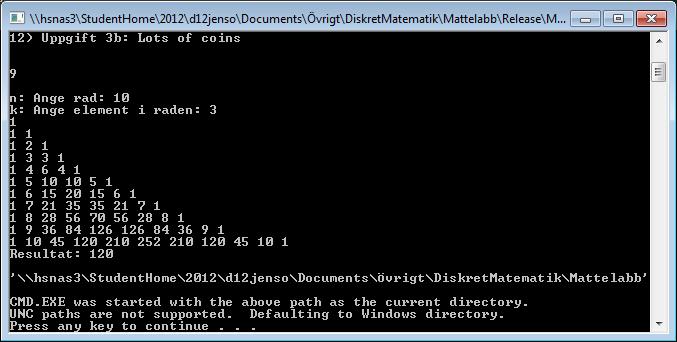
put rowVector in pascalVector

}

return pascalVector[n][k]

}

Funktionen ritar ut Pascals triangel så långt ner som behövs för att nå den önskade raden och returnerar elementet på angiven indexplats. I exemplet blir alltså element 3 det fjärde elementet i raden.



b)

Utveckla (x + y)^n

#### Expand

string x, y

integer n

expansion( x, y, n )

{

for ( 0 < i <= n )

{

integer k = binom\_pascal( n, i )

print: k "(" x ")^" (n – i) " (" y ")^" i

if (i ≠ n)

{

print " + "

}

}

}

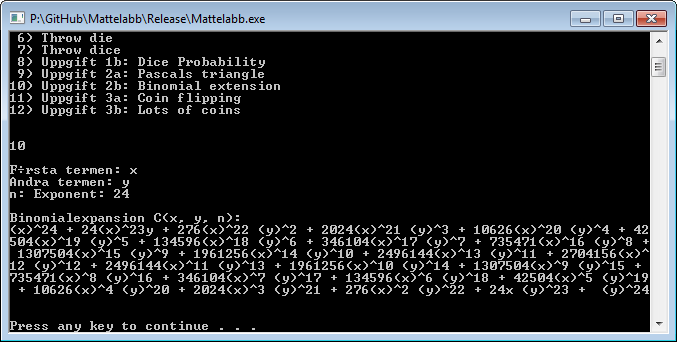
Note: k, i och (n-i) skrivs inte ut om de har värdet 1.

x och y skrivs inte ut om de har värdet 0.

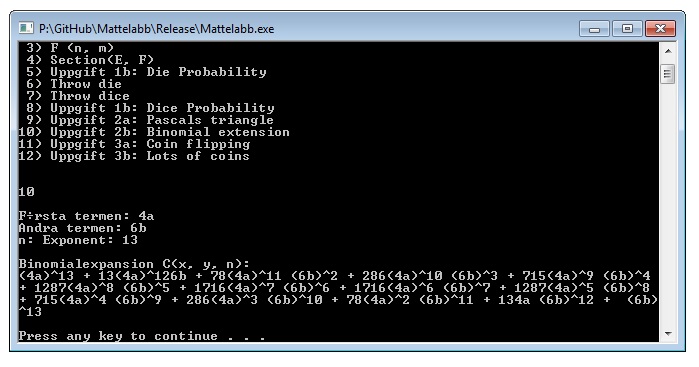
Koefficienterna för varje term fås av binomialsatsen.

c) Utveckla

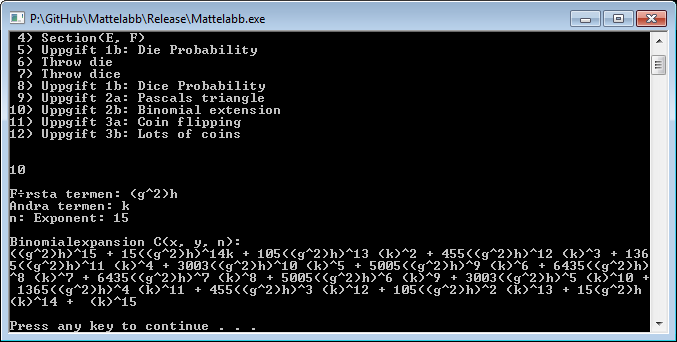
i.



ii.



iii.



### Uppgift 3: Binomialfördelning

a)

krona: 1

klave: 0

#### Kasta mynt

float p

integer kasta\_mynt( p )

{

integer probability = p \* 1000

integer result = randomValue % 1000

if (result < probability)

{

return 1

}

else

{

return 0

}

}

Eftersom det bara finns två möjliga utfall, krona och klave, måste varje resultat som inte är krona vara klave. Det ger att om sannolikheten för krona är p, blir sannolikheten för klave q = 1 – p då de kombinerade sannolikheterna för alla möjliga utfall alltid är ett.

b)

#### Mynt\_experiment

integer n, k

float p

boolean mynt\_experiment( n, k, p )

{

integer count = 0

for ( 0 < i < n )

{

count = ( count + kasta\_mynt( p ) )

}

return true if ( count = k )

}

#### Print\_experiment

integer n

float p

print\_experiment( n, p )

{

integer x = 100000000

for ( 0 < k <= n )

{

integer h = 0

for ( 0 < i < x )

{

if ( mynt\_experiment( n, k, p ) ) returns 1

{

h = h + 1;

}

}

print "k: " k

print "h/x: " h/x

}

}

Syftet var att undersöka sannolikheten för att få krona exakt k gånger när ett mynt kastas n gånger och sannolikheten p för krona i varje enskilt kast är 0.3. Experimentet utfördes 10^8 gånger för varje k<n, för n= 2,3,4,5,10.

n: gånger

k: önskat antal krona

p: sannolikhet för krona

h: antal lyckade experiment

x: antal gånger experimentet utförs

Experimentet utfördes med x=100000000, d.v.s. 10^8 gånger

bool experiment( n, k, p )

h/x för k = 0-n, p = 0.3, n = 2,3,4,5,10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | N = 2 | N=3 | N=4 | N=5 | N=10 |
| K=0 | 0.487056 | 0.339862 | 0.237234 | 0.165521 | 0.0273878 |
| K=1 | 0.421705 | 0.441535 | 0.410788 | 0.358352 | 0.118642 |
| K=2 | 0.0912536 | 0.191092 | 0.266805 | 0.310188 | 0.231196 |
| K=3 | - | 0.0275328 | 0.0769986 | 0.134343 | 0.266817 |
| K=4 | - | - | 0.00833216 | 0.0291038 | 0.202152 |
| K=5 | - | - | - | 0.00251081 | 0.104976 |
| K=6 | - | - | - | - | 0.0378986 |
| K=7 | - | - | - | - | 0.00936076 |
| K=8 | - | - | - | - | 0.00151394 |
| K=9 | - | - | - | - | 0.00014641 |
| K=10 | - | - | - | - | 6.01\*10^-6 |

Eftersom sannolikheten för att få krona är lite mindre än hälften så stor som sannolikheten för att få krona, ( p = 0.3 ger 1 – p = q = 0.7 ), blir det inte en normalfördelning av resultaten. Det som fås är en tvåpunktsfördelning, mer specifikt en Bernoullifördelning då p ≠ q.

c)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | N = 2 | N=3 | N=4 | N=5 | N=10 |
| K=0 | 0.49 | 0.343 | 0.2401 | 0.16807 | 0.0282475249 |
| K=1 | 0.42 | 0.441 | 0.4116 | 0.36015 | 0.121060821 |
| K=2 | 0.09 | 0.189 | 0.2646 | 0.3087 | 0.2334744405 |
| K=3 | - | 0.027 | 0.0756 | 0.1323 | 0.266827932 |
| K=4 |  | - | 0.0081 | 0.02835 | 0.200120949 |
| K=5 |  |  | - | 0.00243 | 0.102919345 |
| K=6 |  |  |  | - | 0.036756909 |
| K=7 |  |  |  |  | 0.009001692 |
| K=8 |  |  |  |  | 0.001446701 |
| K=9 |  |  |  |  | 0.000137781 |
| K=10 |  |  |  |  | 5.9049\*10^-6 |

Värdena från experimenten motsvarar ungefär de som fås av formeln men skulle bli mer exakta med ett högre antal utförda experiment.

d)

E[X] = np

p = 0.3

Väntevärdet innebär det genomsnittliga värdet av experimentet. För n = 10 innebär det till exempel att om en slant singlas tio gånger och sannolikheten vid varje enskilt kast för att få krona är 0.3, är det sannolikt att det totala värdet på alla kast blir 3, det vill säga att av tio kast är det sannolikt att precis tre av dessa är krona.

# Appendix:

#### Main.cpp

#include <iostream>

#include <math.h>

#include <vector>

#include <stdlib.h>

#include <ctime>

#include "dice.h"

#include "binom.h"

#include "coin.h"

using namespace std;

int inputN();

int inputM();

int inputX();

int main()

{

Reset:

Dice::generate\_seed();

Coin::generate\_seed();

int input = 0;

int n = 0;

int m = 0;

int x = 0;

int y = 0;

int k = 0;

float p = 0;

string X;

string Y;

cout << "\tAvailable functions:\n\n\n";

cout << " 1) Uppgift 1a: Omega (n, m)\n";

cout << " 2) E (n, m)\n";

cout << " 3) F (n, m)\n";

cout << " 4) Section(E, F)\n";

cout << " 5) Uppgift 1a: Die Probability\n";

cout << " 6) Throw die\n";

cout << " 7) Throw dice\n";

cout << " 8) Uppgift 1b: Dice Probability\n";

cout << " 9) Uppgift 2a: Pascals triangle\n";

cout << "10) Uppgift 2b: Binomial extension\n";

cout << "11) Uppgift 3a: Coin flipping\n";

cout << "12) Uppgift 3b: Lots of coins\n";

cout << endl << endl;

cin >> input;

cout << endl;

switch (input)

{

case 1:

n = inputN();

m = inputM();

cout << endl;

Dice::printSet(Dice::Omega(n, m));

break;

case 2:

n = inputN();

m = inputM();

cout << endl;

Dice::Omega(n,m);

Dice::printSet(Dice::E(n, m));

break;

case 3:

n = inputN();

m = inputM();

cout << endl;

Dice::Omega(n,m);

Dice::printSet(Dice::F(n, m));

break;

case 4:

n = inputN();

m = inputM();

cout << endl;

Dice::Omega(n,m);

Dice::printSet(Dice::section(Dice::E(n, m), Dice::F(n, m)));

break;

case 5:

n = inputN();

m = inputM();

cout << endl;

Dice::Omega(n,m);

Dice::probability(n, m);

break;

case 6:

m = inputM();

cout << endl;

Dice::kasta\_tarning(m);

break;

case 7:

n = inputN();

m = inputM();

cout << endl;

Dice::kasta\_tarningar(n, m);

break;

case 8:

n = inputN();

m = inputM();

x = inputX();

cout << endl;

Dice::oneBprobability(n, m, x);

break;

case 9:

cout << "n: Ange rad: ";

cin >> n;

cout << "k: Ange element i raden: ";

cin >> k;

cout << "Resultat: " << Binom::binom\_pascal(n, k, 1); // 1 = print triangle

break;

case 10:

cout << "Första termen: ";

cin >> X;

cout << "Andra termen: ";

cin >> Y;

cout << "n: Exponent: ";

cin >> n;

cout << "\nBinomialexpansion C(x, y, n): \n";

Binom::extension(X, Y, n);

break;

case 11:

cout << "p: Ange sannolikhet (flyttal 0-1) ";

cin >> p;

cout << "Result: " << Coin::kasta\_mynt(p);

break;

case 12:

cout << "n: Ange antal myntkast: ";

cin >> n;

cout << "p: sannolikhet för krona: ";

cin >> p;

Coin::print\_experiment(n, p);

break;

default:

goto Reset;

break;

}

cout << "\n\n";

system("PAUSE");

system("CLS");

goto Reset;

system("PAUSE");

return 0;

}

int inputN() // Number of dice/throws

{

int n = 0;

cout << "Number of dice/throws: ";

cin >> n;

return n;

}

int inputM() // Number of sides on the die

{

int m = 0;

cout << "Number of die sides: ";

cin >> m;

return m;

}

int inputX() // Number of experimental iterations

{

int x = 0;

cout << "Number of experiments: ";

cin >> x;

return x;

}

#### Dice.h

#ifndef \_DICE\_

#define \_DICE\_

#include <iostream>

#include <math.h>

#include <vector>

#include <stdlib.h>

#include <ctime>

using namespace std;

typedef vector<vector<int>> vec2;

class Dice

{

public:

static void generate\_seed();

static vec2& Omega(int n, int m);

static vec2 E(int n, int m);

static vec2 F(int n, int m);

static void printSet(vec2 a);

static vec2 section(vec2& a, vec2& b);

static void probability(int n, int m);

static int kasta\_tarning(int m);

static vector<int> kasta\_tarningar(int n, int m);

static void oneBprobability(int n, int m, int x);

private:

static vec2 omega;

};

#endif // \_DICE\_

#### Dice.cpp

#include "dice.h"

vec2 Dice::omega;

void Dice::generate\_seed()

{

srand(time(NULL));

}

vec2& Dice::Omega(int n, int m) //n = no throws, m = no sides

{

omega.clear();

cout << "Executing function Omega\n\n";

vector <int> list;

//vector < vector <int> > metalist;

int permutations = pow(m, n);

omega.reserve(permutations);

cout << "Number of possible combinations = " << permutations << endl << endl;

// For each loop, create a vector

for (int i = 0; i < permutations; i++)

{

// Clear list

list.clear();

// For each loop fill a vector

for (int j = n - 1; j >= 0; j--)

{

int power = pow(m, j);

// Det nya elementet fås av en formel beroende på antalet sidor,

// dess position i arrayen samt antalet element arrayen håller

int newElement = ((i / power) % m) + 1;

list.push\_back(newElement);

}

// Put the vector in the vector-vector-reference

omega.push\_back(list);

}

// What's our vector, Victor?

return omega;

}

vec2 Dice::E(int n, int m)

{

cout << "Executing function E\n\n";

vec2 metalist;

for (int i = 0; i < omega.size(); i++)

{

vector< int >& list = omega[i];

bool allValues = true;

for (int die = 1; die <= m; die++)

{

bool exists = false;

for (int k = 0; k < list.size(); k++)

{

if (list[k] == die)

{

exists = true;

}

}

if (exists == false)

{

allValues = false;

break;

}

}

if (allValues == true)

{

metalist.push\_back(list);

}

}

return metalist;

}

vec2 Dice::F(int n, int m)

{

cout << "Executing function F\n\n";

vec2 metalist;

for (int i = 0; i < omega.size(); i++)

{

bool lessThan = false;

for (int j = 1; j < omega[i].size(); j++)

{

if (omega[i][j] < omega[i][j - 1])

{

lessThan = true;

break;

}

}

if (!lessThan)

{

metalist.push\_back(omega[i]);

}

}

return metalist;

}

void Dice::printSet(vec2 inputSet)

{

cout << "InputSet: "<<inputSet.size() << endl;

for (int i = 0; i < inputSet.size(); i++)

{

vector <int> list = inputSet[i];

for (int j = 0; j < list.size(); j++)

{

cout << list[j] << ", ";

}

cout << endl;

}

}

vec2 Dice::section(vec2& A, vec2& B)

{

cout << "Executing function Section\n\n";

vec2 sect;

//sect.reserve(A.size()\*A[0].size());

for (int i = 0; i < A.size(); i++)

{

for (int j = 0; j < B.size(); j++)

{

if (A[i] == B[j])

{

sect.push\_back(A[i]);

break;

}

}

}

return sect;

}

void Dice::probability(int n, int m)

{

cout << "Executing function Probability\n\n";

vec2 e = E(n, m);

vec2 f = F(n, m);

vec2 s = section(e, f);

cout << "P(E): " << float(e.size()) / float(omega.size()) << endl;

cout << "P(F): " << float(f.size()) / float(omega.size()) << endl;

cout << "P(E|F): " << float(s.size()) / float(f.size()) << endl;

}

int Dice::kasta\_tarning(int m) // m sidor

{

if (m < 2)

{

return NULL;

}

int value = rand() % m + 1;

return value;

}

vector<int> Dice::kasta\_tarningar(int n, int m)

{

vector<int> values;

for (int i = 0; i < n; i++)

{

values.push\_back(kasta\_tarning(m));

}

return values;

}

void Dice::oneBprobability(int n, int m, int x)

{

vec2 values;

for (int i = 0; i < x; i++)

{

vector<int> tempValues = kasta\_tarningar(n, m);

values.push\_back(tempValues);

}

omega = values;

cout << "E: " << E(n, m).size() << endl;

int b = F(n, m).size();

cout << "F: " << b << endl;

int a = section(E(n, m), F(n, m)).size();

if (b != 0)

{

float probability = (float) a / (float) b;

cout << a << endl;

cout << b << endl;

cout << n << " " << m << "-sided dice " << "thrown " << x << " times\n gives P(E|F): " << probability << endl;

}

else

{

cout << a << endl;

cout << b << endl;

cout << "Don't divide by zero, stupid." << endl;

}

}

#### Binom.h

#ifndef \_BINOM\_

#define \_BINOM\_

#include <iostream>

#include <math.h>

#include <vector>

#include <memory>

#include <string>

using namespace std;

typedef vector<int> littlePascalVector;

class Binom

{

public:

static int binom\_pascal(int n, int k, bool p);

static void extension(string x, string y, int n);

};

#endif // \_BINOM\_

#### Binom.cpp

#include "binom.h"

int Binom::binom\_pascal(int n, int k, bool p)

{

if (k > n)

{

return 0;

}

vector<vector<int>\*> pascalVector;

for (int i = 0; i <= n; i++) // raden

{

vector<int>\* v= new vector<int>();

for (int j = 0; j <= i; j++) // elementet i raden

{

if (j == 0 || j == i)

{

v->push\_back(1);

if(p) cout << "1 ";

}

else

{

int firstValue = pascalVector.at(i - 1)->at(j - 1);

int secondValue = pascalVector.at(i - 1)->at(j);

if(p) cout << firstValue + secondValue <<" ";

v->push\_back(firstValue+secondValue);

}

}

pascalVector.push\_back(v);

if(p) cout << endl;

}

return pascalVector.at(n)->at(k);

}

void Binom::extension(string x, string y, int n)

{

for (int i = 0; i <= n; i++)

{

unsigned long int k = binom\_pascal(n, i, 0);

if (k > 1)

{

cout << k;

}

if ((n - i) != 1)

{

if ((n - i) != 0) cout << "(" << x << ")^" << n - i;

}

else

{

cout << x;

}

if (i != 1)

{

if (i != 0) cout << " (" << y << ")^" << i;

}

else

{

cout << y;

}

if (i != n)

{

cout << " + ";

}

}

}

#### Coin.h

#ifndef \_COIN\_

#define \_COIN\_

#include <stdlib.h>

#include <ctime>

#include <iostream>

#include <cassert>

using namespace std;

class Coin

{

public:

static void generate\_seed();

static int kasta\_mynt(float p);

static bool experiment(int n, int k, float p);

static void print\_experiment(int n, float p);

};

#endif //\_COIN\_

#### Coin.cpp

#include "coin.h"

void Coin::generate\_seed()

{

srand(time(NULL));

}

int Coin::kasta\_mynt(float p)

{

assert(p >= 0 && p <= 1);

int result = 0;

int probability = p \* 100000;

result = rand() % 100000;

assert(result >= 0 && result < 100000);

if (result < probability)

{

return 1;

}

else

{

return 0;

}

}

bool Coin::experiment(int n, int k, float p)

{

int count = 0;

for (int i = 0; i < n; i++)

{

count += kasta\_mynt(p);

}

return (count == k);

}

void Coin::print\_experiment( int n, float p)

{

int x = 100000000;

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

{

int h = 0;

for (int i = 0; i < x; i++)

{

if (experiment( n, k, p))

{

h++;

}

}

cout << "k: " << k << " h/x: " << (float)h/(float)x << endl;

}

}