Листинг программного кода для исследования архитектуры TPM

UPerceptron.h

```
#ifndef UPerceptronH
#define UPerceptronH
#include <stdlib.h>

class Perceptron
{
  int *Weights;
  int *Inputs;
  int Output;
  void RandomWeights();
public:
  Perceptron();
  ~Perceptron();
  int GetOutput(int *AInputs);
  void AktualizeWeights(int OutputTPM);
  int Distance(const Perceptron &p);
};
#endif
//---------------------------------------------------------------------------
```

UPerceptron.cpp

```
#pragma hdrstop
#include "UPerceptron.h"
#include "UTPM.h"
#pragma package(smart_init)

Perceptron::Perceptron()
{
  Weights = new int[N];
  RandomWeights();
}
Perceptron::~Perceptron()
{
  delete []Weights;
}
int Perceptron::GetOutput(int *AInputs)
{
  Inputs = AInputs;
```

```
  Output = 0;
  for (int i = 0; i < N ; i++)
    Output += Inputs[i]*Weights[i];
  if (Output >= 0) Output = 1;
    else
  Output = -1;
  return Output;
}
void Perceptron::RandomWeights()
{
  for (int i = 0; i < N; i++)
    Weights[i] = L - random(2*L);
}
void Perceptron::AktualizeWeights(int OutputTPM)
{
if (OutputTPM == Output)
for (int i = 0; i < N; i++)
  {
   Weights[i] += Output*Inputs[i];

   if (Weights[i] > L) Weights[i] = L;
     else
   if (Weights[i] < -L) Weights[i] = -L;
  }
}
int Perceptron::Distance(const Perceptron &P)
{
  int Result = 0;
   for (int i = 0; i < N; i++)
    Result += abs(Weights[i] - P.Weights[i]);
  return Result;
}
//---------------------------------------------------------------------------
```

UTPM.h

```
#ifndef UTPMH
#define UTPMH
#include "UPerceptron.h"
const int K = 3;
const int L = 3;
const int N = 16;
```

```
class TPM
{
  Perceptron Perceptrons[K];
  int Output;
public:
  int GetOutput(int AInputs[K][N]);
  void Synchronize();
  int Distance(const TPM &ATPM);
  void ModifyOutput(int AOutput){Output = AOutput;};
};
#endif
//---------------------------------------------------------------------------
```

UTPM.cpp

```
#pragma hdrstop
#include "UTPM.h"
#pragma package(smart_init)
int TPM::GetOutput(int AInputs[K][N])
{
  Output = 1;
  for (int i = 0; i < K ; i++)
    Output *= Perceptrons[i].GetOutput(AInputs[i]);

  return Output;
}
void TPM::Synchronize()
{
  for (int i = 0; i < K; i++)
    Perceptrons[i].AktualizeWeights(Output);
}
int TPM::Distance(const TPM &ATPM)
{
  int Result = 0;
   for (int i = 0; i < K; i++)
    Result += Perceptrons[i].Distance(ATPM.Perceptrons[i]);
  return Result;
}
```